EXHIBIT H

DEEP LEARNING

Deep Neural Network: The 3 Popular Types (MLP, CNN and RNN)

Gaudenz Boesch

About

Viso Suite is the no-code computer vision platform to build, deploy and scale any application 10x faster.

Follow the blog

This article will explain the differences between the three types of neural networks and cover the basics of Deep Neural Networks. Such deep neural networks (DNNs) have recently demonstrated impressive performance in complex machine learning tasks such as image classification or text and speech recognition.

In particular, we will cover the following neural network types:

- Multi-Layer Perceptrons (MLP)

Follow the blog

Contents

- Multi-Layer Perceptrons (MLP)
- Convolutional Neural Networks (CNN)
- Recurrent Neural Networks (RNN)

What Is a Deep Neural Network?

Machine learning techniques have been widely applied in various areas such as pattern recognition, natural language processing, and computational learning. During the past decades, machine learning has brought enormous influence on our daily life with examples including efficient web search, self-driving systems, computer vision, and optical character recognition (OCR).

Especially, deep neural network models have become a powerful tool for machine learning and artificial intelligence. A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers. Note that the terms ANN vs. DNN are often incorrectly confused or used interchangeably.

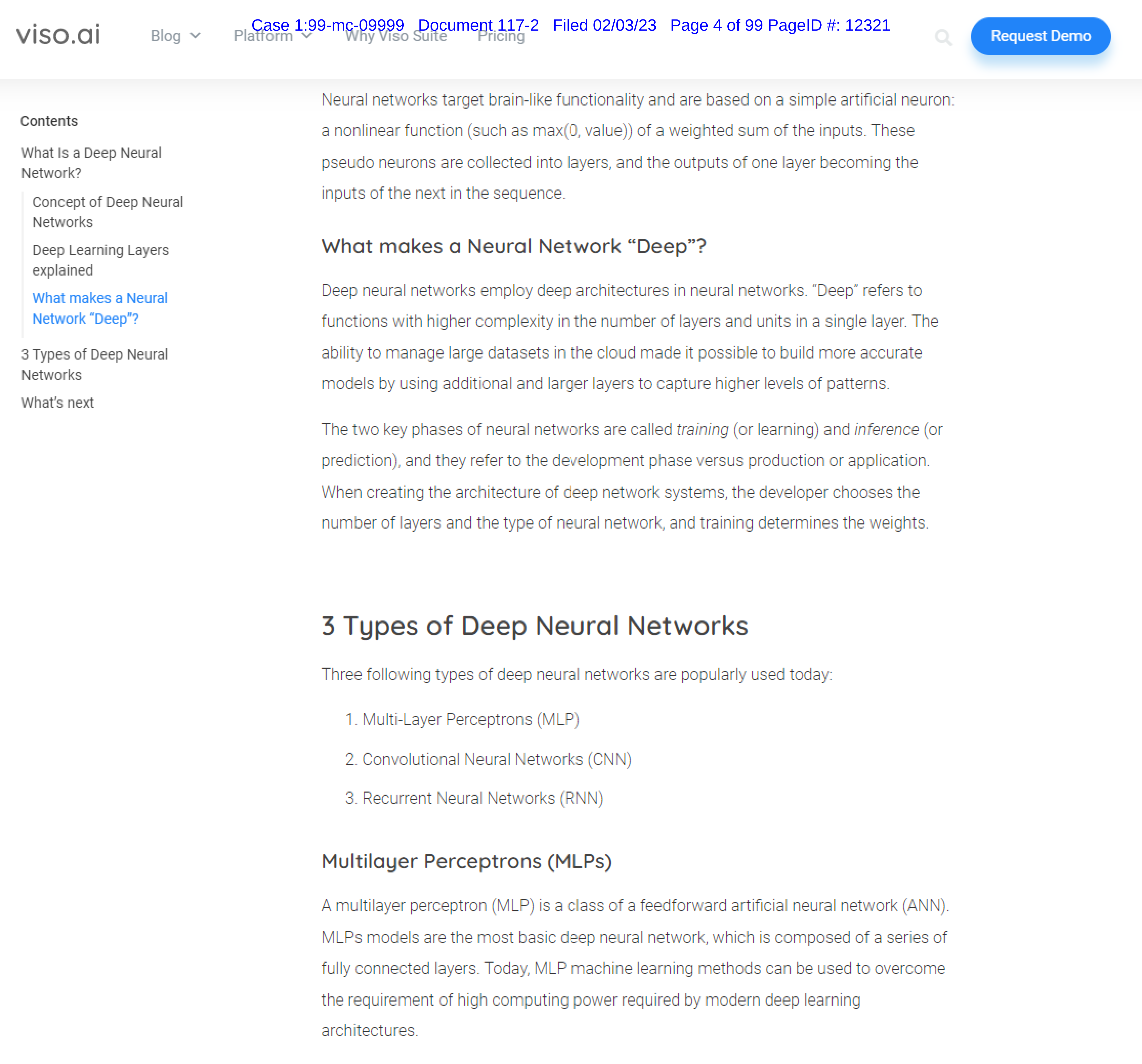
The success of deep neural networks has led to breakthroughs such as reducing word error rates in speech recognition by 30% over traditional approaches (the biggest gain in 20 years) or drastically cutting the error rate in an image recognition competition since 2011 (from 26% to 3.5% while humans achieve 5%).

Concept of Deep Neural Networks

Deep neural network models were originally inspired by neurobiology. On a high level, a biological neuron receives multiple signals through the synapses contacting its dendrites and sends a single stream of action potentials out through its axon. The complexity of multiple inputs is reduced by categorizing its input patterns. Inspired by this intuition, artificial neural network models are composed of *units* that combine multiple inputs and produce a single output.

Deep Learning Layers explained

Neural networks target brain-like functionality and are based on a simple artificial neuron:



Contents

Neural networks target brain-like functionality and are based on a simple artificial neuron: a nonlinear function (such as max(0, value)) of a weighted sum of the inputs. These pseudo neurons are collected into layers, and the outputs of one layer becoming the inputs of the next in the sequence.

What makes a Neural Network “Deep”?

Deep neural networks employ deep architectures in neural networks. “Deep” refers to functions with higher complexity in the number of layers and units in a single layer. The ability to manage large datasets in the cloud made it possible to build more accurate models by using additional and larger layers to capture higher levels of patterns.

The two key phases of neural networks are called *training* (or learning) and *inference* (or prediction), and they refer to the development phase versus production or application. When creating the architecture of deep network systems, the developer chooses the number of layers and the type of neural network, and training determines the weights.

3 Types of Deep Neural Networks

Three following types of deep neural networks are popularly used today:

1. Multi-Layer Perceptrons (MLP)
2. Convolutional Neural Networks (CNN)
3. Recurrent Neural Networks (RNN)

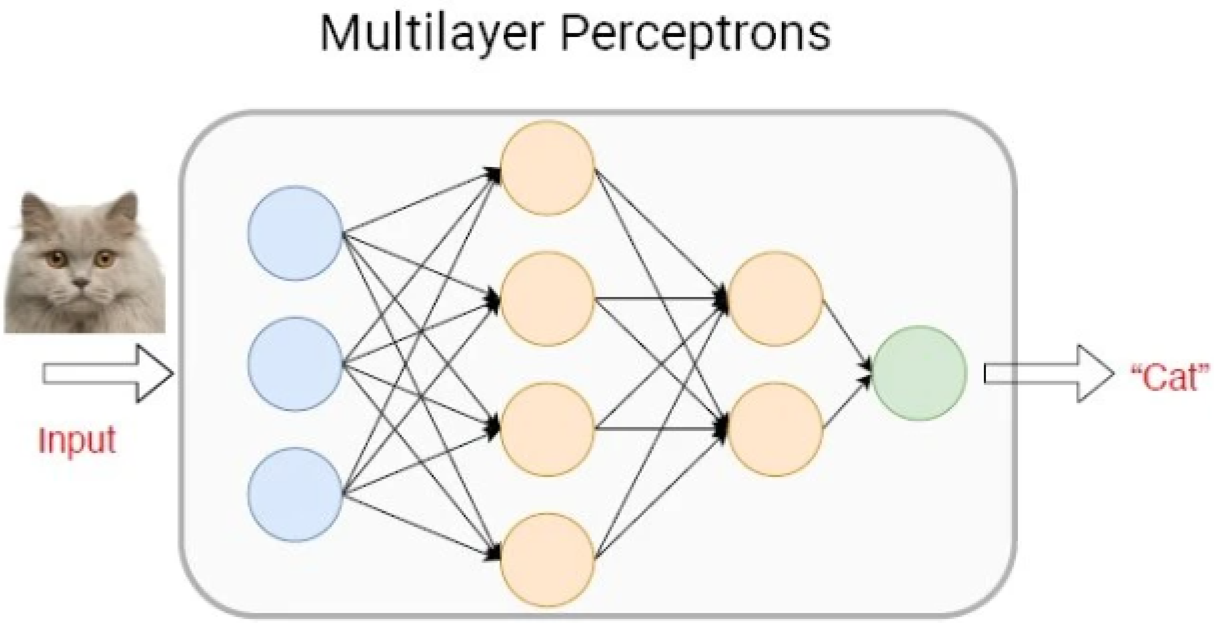
Multilayer Perceptrons (MLPs)

A multilayer perceptron (MLP) is a class of a feedforward artificial neural network (ANN). MLPs models are the most basic deep neural network, which is composed of a series of fully connected layers. Today, MLP machine learning methods can be used to overcome the requirement of high computing power required by modern deep learning architectures.

Contents

the requirement of high computing power required by modern deep learning architectures.

Each new layer is a set of nonlinear functions of a weighted sum of all outputs (fully connected) from the prior one.

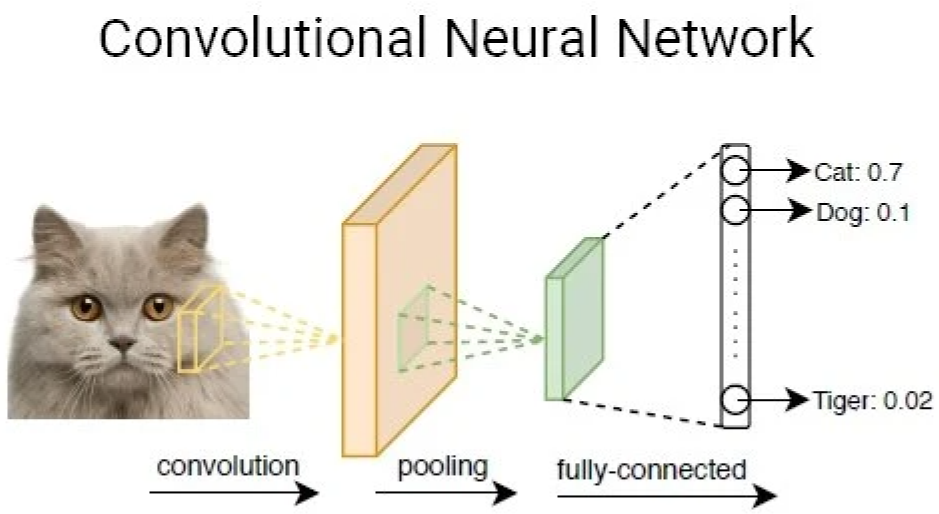


Concept of Multilayer Perceptrons (MLP)

Convolutional Neural Network (CNN)

A convolutional neural network (CNN, or ConvNet) is another class of deep neural networks. CNNs are most commonly employed in computer vision. Given a series of images or videos from the real world, with the utilization of CNN, the AI system learns to automatically extract the features of these inputs to complete a specific task, e.g., image classification, face authentication, and image semantic segmentation.

Different from fully connected layers in MLPs, in CNN models, one or multiple convolution layers extract the simple features from input by executing convolution operations. Each layer is a set of nonlinear functions of weighted sums at different coordinates of spatially nearby subsets of outputs from the prior layer, which allows the convolution layers extract the simple features from input by executing convolution operations. Each layer is a set of nonlinear functions of weighted sums at different coordinates of spatially nearby subsets of outputs from the prior layer, which allows the weights to be reused.



Concept of a Convolution Neural Network (CNN)

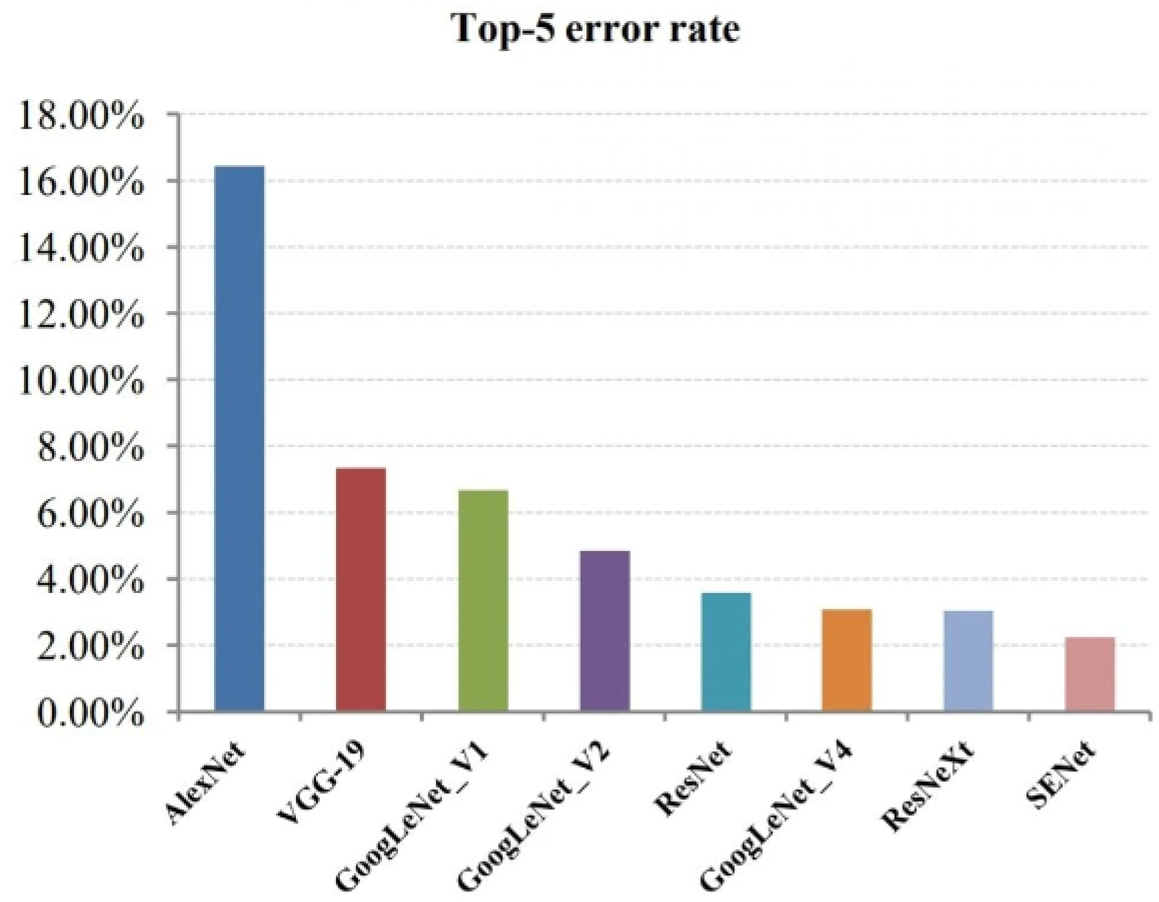
Applying various convolutional filters, CNN machine learning models can capture the high-level representation of the input data, making CNN techniques widely popular in computer vision tasks. Convolutional neural network example applications include image classification (e.g., AlexNet, VGG network, ResNet, MobileNet) and object detection (e.g., Fast R-CNN, Mask R-CNN, YOLO, SSD).

- AlexNet. For image classification, as the first CNN neural network to win the ImageNet Challenge in 2012, AlexNet consists of five convolution layers and three fully connected layers. Thus, AlexNet requires 61 million weights and 724 million MACs (multiply-add computation) to classify the image with a size of 227×227.
- VGG-16. To achieve higher accuracy, VGG-16 is trained to a deeper structure of 16 layers consisting of 13 convolution layers and three fully connected layers, requiring 138 million weights and 15.5G MACs to classify the image with a size of 224×224.
- GoogleNet. To improve accuracy while reducing the computation of DNN

224×224.

- GoogleNet. To improve accuracy while reducing the computation of DNN inference, GoogleNet introduces an inception module composed of different sized filters. As a result, GoogleNet achieves a better accuracy performance than VGG-16 while only requiring seven million weights and 1.43G MACs to process the image with the same size.
- ResNet. ResNet, the state-of-the-art effort, uses the "shortcut" structure to reach a human-level accuracy with a top-5 error rate below 5%. In addition, the "shortcut" module is used to solve the gradient vanishing problem during the training process, making it possible to train a DNN model with a deeper structure.

The performance of popular CNNs applied for AI vision tasks gradually increased over the years, surpassing human vision (5% error rate in the chart below).



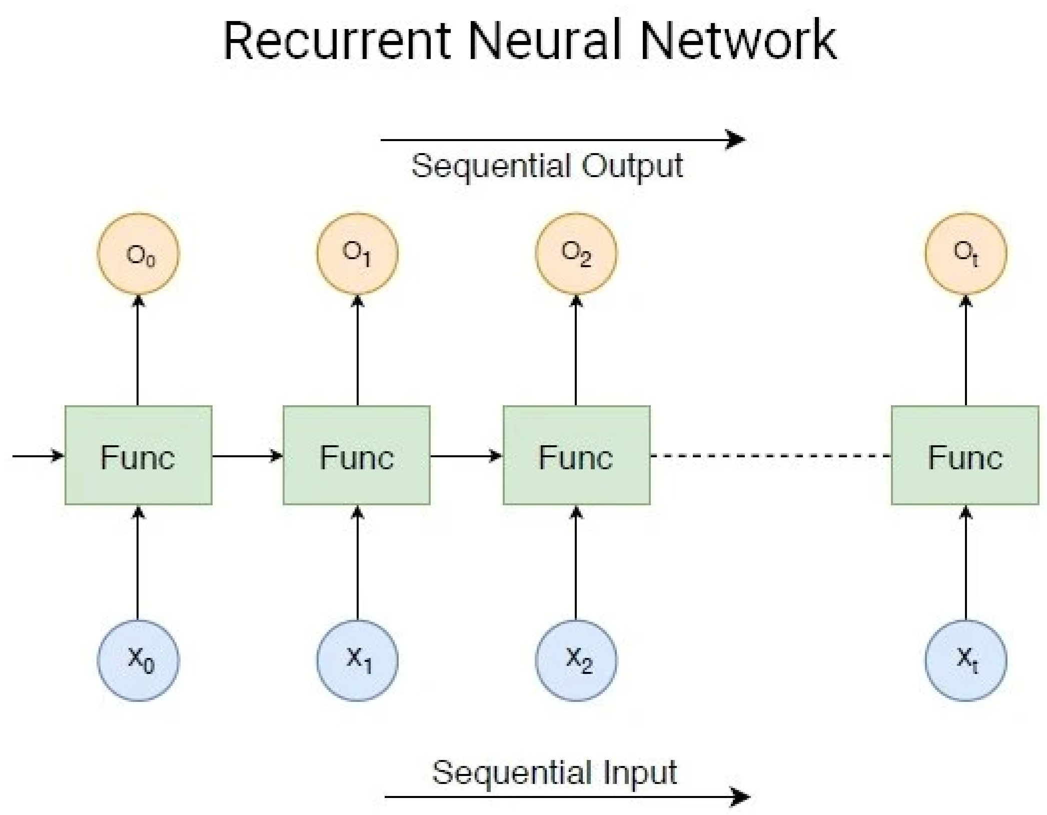
Performance of current popular Deep Neural Networks on ImageNet. Humans achieve an error rate of 5%. – Source

Performance of current popular Deep Neural Networks on ImageNet. Humans achieve an error rate of 5%. – Source

Recurrent Neural Network (RNN)

A recurrent neural network (RNN) is another class of artificial neural networks that use sequential data feeding. RNNs have been developed to address the time-series problem of sequential input data.

The input of RNN consists of the current input and the previous samples. Therefore, the connections between nodes form a directed graph along a temporal sequence. Furthermore, each neuron in an RNN owns an internal memory that keeps the information of the computation from the previous samples.



Concept of a Recurrent Neural Network (RNN)

RNN models are widely used in Natural Language Processing (NLP) due to the

Contents

RNN models are widely used in Natural Language Processing (NLP) due to the superiority of processing the data with an input length that is not fixed. The task of the AI here is to build a system that can comprehend natural language spoken by humans, e.g., natural language modeling, word embedding, and machine translation.

In RNNs, each subsequent layer is a collection of nonlinear functions of weighted sums of outputs and the previous state. Thus, the basic unit of RNN is called "cell", and each cell consists of layers and a series of cells that enables the sequential processing of recurrent neural network models.

What's next

Deep neural networks excel at finding hierarchical representations that solve complex tasks with large datasets. Each category and architecture of deep network systems provide task-specific characteristics. To learn about using deep neural networks in state-of-the-art image recognition, check out our article Image Recognition today: A Comprehensive Guide.

At the Viso Computer Vison Blog We also cover other popular topics related to computer vision and deep learning technologies. We recommend you explore the following topics:

- Read about the difference between CNN and ANN.
- An easy-to-understand guide to Deep Reinforcement Learning.
- Read an introduction to Self-Supervised Learning.
- Learn about the difference between Deep Learning vs. Machine Learning.

RELATED ARTICLES SHOW MORE ›

RELATED ARTICLES

SHOW MORE ›

Animal Monitoring with Computer Vision – Case Study

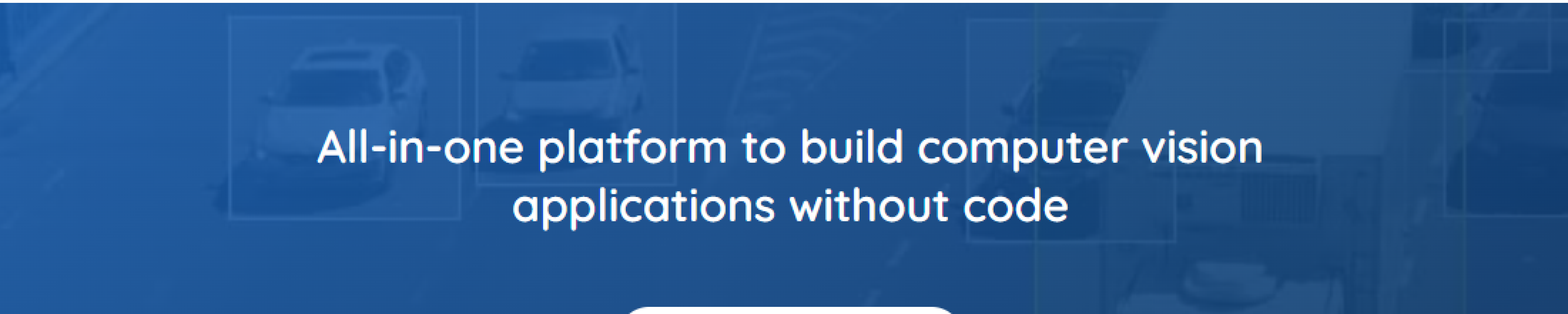
Animal monitoring with computer vision can surpass human accuracy. It offers the possibility of non-intrusive animal inspection.

Read More »

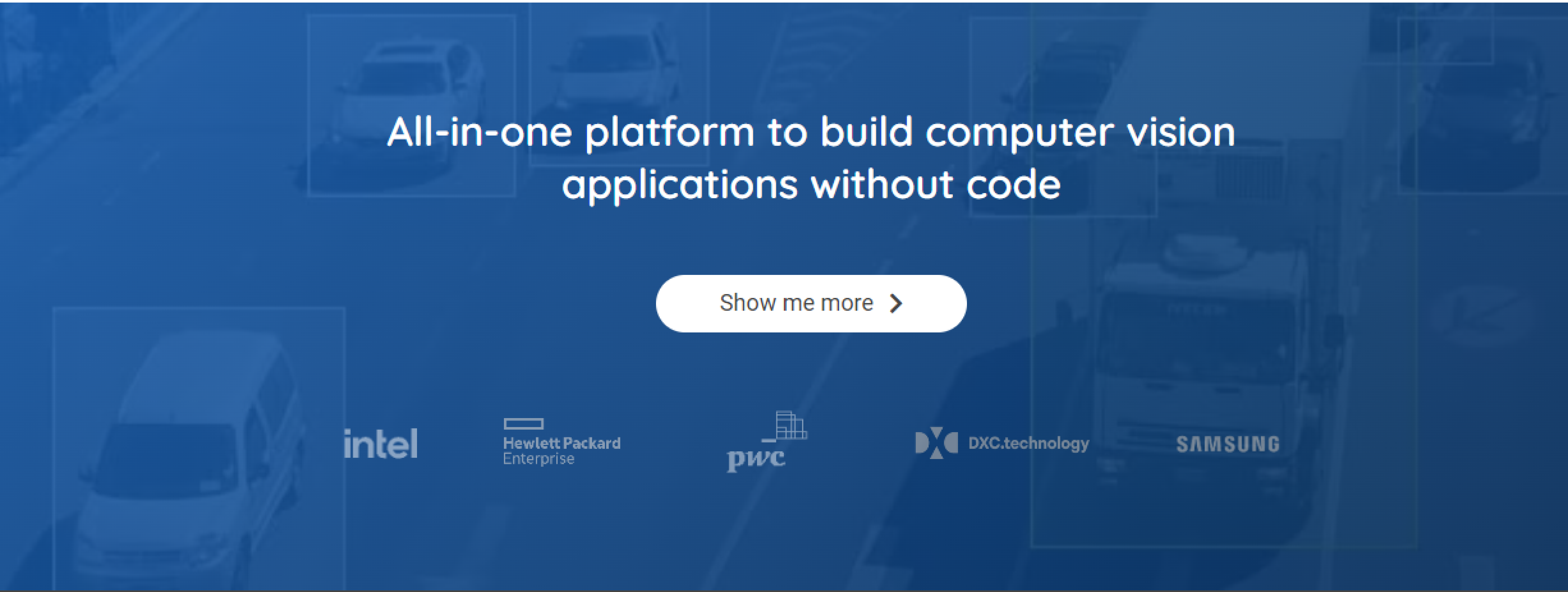
viso.ai Joins NVIDIA Inception Program

viso.ai joins the NVIDIA Inception Program for AI Startups to accelerate the AI ecosystem across the globe. A platform for AI vision.

Read More »



All-in-one platform to build computer vision applications without code



All-in-one platform to build computer vision applications without code

Show me more ›

intel Hewlett Packard Enterprise pwc DXC.technology SAMSUNG

viso.ai

Product
- Overview
- Evaluation Guide
- Feature Index
- Academy
- Security
- Privacy
- Solutions
- Pricing

Features
- Computer Vision
- Visual Programming
- Cloud Workspace
- Analytics Dashboard
- Device Management
- End-to-End Suite

Industries
- Agriculture
- Healthcare
- Manufacturing
- Retail
- Security
- Smart City
- Technology
- Transportation

Resources
- Blog
- Learn
- Evaluation
- Support
- Whitepaper

About
- Company
- Careers
- Terms
- Contact

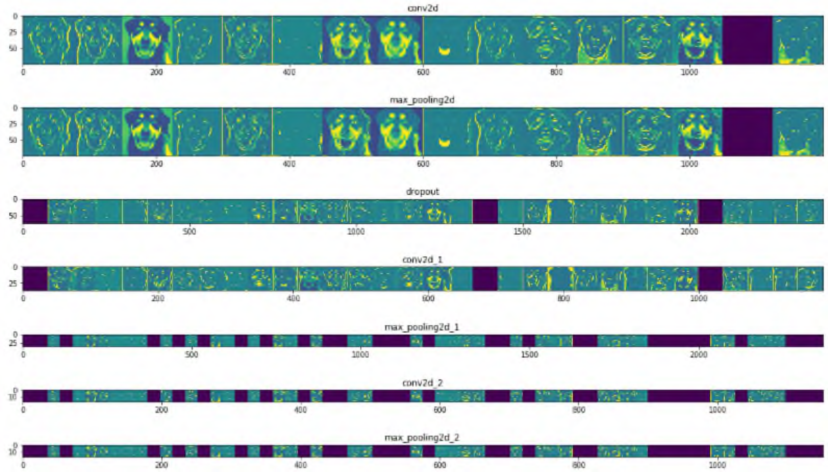
EXHIBIT I

Published in Towards Data Science

Renu Khandelwal
May 18, 2020 · 8 min read · Member-only · Listen

Convolutional Neural Network: Feature Map and Filter Visualization

Learn how Convolutional Neural Networks understand images.



In this article, we will visualize the intermediate feature representations across different CNN layers to understand what happens inside CNN's to classify images.

Prerequisites:

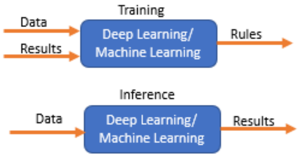
Convolutional Neural Network Basics,

Building Powerful Image Classification Convolutional Neural Network using Keras

Building powerful image classification CNN using Keras

A quick overview of CNN

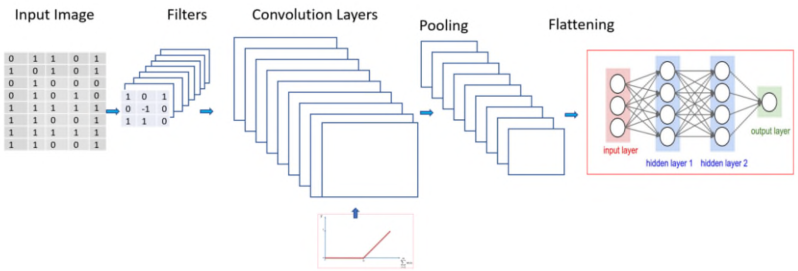
Supervised Deep Learning and Machine Learning take data and results as an input during training to generate the rules or data patterns. Understanding of data patterns or rules generated by the model helps us understand how the results were derived from the input data.


Training
Data
Results
Deep Learning/ Machine Learning
Rules
Inference
Data
Deep Learning/ Machine Learning
Results

Training: Convolutional neural network takes a two-dimensional image and the class of the image, like a cat or a dog as an input. As a result of the training, we get trained weights, which are the data patterns or rules extracted from the images.

Inference or Prediction: Image will be the only input passed to the trained model, and the trained model will output the class of the image. The class of the image will be based on the learned data patterns during the training.

CNN Architecture



0	1	1	0	1
1	0	1	0	1
0	1	0	0	0
0	1	0	1	0
1	1	1	1	1
1	1	0	0	1
1	1	1	1	1
1	1	0	0	1

1	0	1
0	-1	0
1	1	0

Apply filters or **feature detectors to the input image** to generate the **feature maps or the activation maps using the Relu activation function.** Feature detectors or filters help identify different features present in an image like edges, vertical lines, horizontal lines, bends, etc.

Pooling is then applied over the feature maps for invariance to translation. Pooling is based on the concept that when we change the input by a small amount, the pooled outputs do not change. We can use min pooling, average pooling, or max pooling. Max pooling provides better performance compared to min or average pooling.

Flatten all the input and pass these flattened inputs to a deep neural network that outputs the class of the object

The class of the image can be binary like a cat or dog, or it can be a multi-class classification like identifying digits or classifying different apparel items.

Neural networks are like a black box, and learned features in a Neural Network are not interpretable. You pass an input image, and the model returns the results.

What if you get an incorrect prediction and would like to figure out why such a decision was made by CNN?

> If only you could visualize the intermediate representation applied across different Convolutional layers in CNN to understand how the model learns. Understanding the working of the model will help to know the reason for incorrect predition that will lead to better fine tuning of the model and explain the decisions

The example used here is a deep CNN model for classifying cats and dogs. Before you dive in to learn to visualize both the filters and the feature maps generated by CNN, you will need to understand some of the critical points about Convolutional layers and the filters applied to them.

Key points about Convolution layers and Filters

- **The depth of a filter in a CNN must match the depth of the input image.** The number of color channels in the filter must remain the same as the input image.
- **Different Conv2D filters are created for each of the three channels** for a color image.
- **Filters for each layer are randomly initialized based on either Normal or Gaussian distribution.**
- **Initial layers of a convolutional network extract high-level features from the image, so use fewer filters.** As we build further deeper layers, we increase the number of filters to twice or thrice the size of the filter of the previous layer.
- **Filters of the deeper layers learn more features but are computationally very intensive.**

Building a Convolutional Neural Network

We build a CNN for classifying dogs and cats and later visualize the feature maps or activation maps and filters applied to generate them on an input image

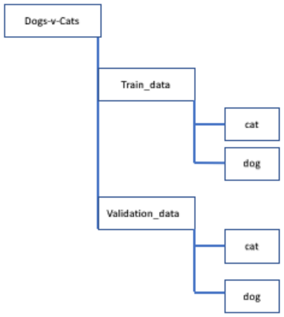
Importing required libraries

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout,
MaxPooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator,
img_to_array, load_img

import os
import numpy as np
import matplotlib.pyplot as plt
```

Creating data for image processing

Dataset can be downloaded from here

We will unzip the file and create the folders as shown below and split the data into the training dataset with 10,000 cats and 10,000 dogs images and validation dataset containing 2500 cats and 2500 dogs images



Set key parameters

```
batch_size = 64
epochs = 50
IMG_HEIGHT = 150
IMG_WIDTH = 150
```

Rescale and Apply different Augmentation to the training image

```
train_image_generator = ImageDataGenerator(
rescale=1./255,
rotation_range=45,
width_shift_range=.15,
height_shift_range=.15,
horizontal_flip=True,
zoom_range=0.3)
```

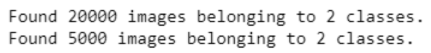
Rescale Validation data

```
validation_image_generator = ImageDataGenerator(rescale=1./255)
```

Generate batches of normalized data for train and validation data set

your data is stored in directories, so use the **flow_from_directory()** method. flow_from_directory() will take the data from the specified path and generates batches of augmented normalized data.

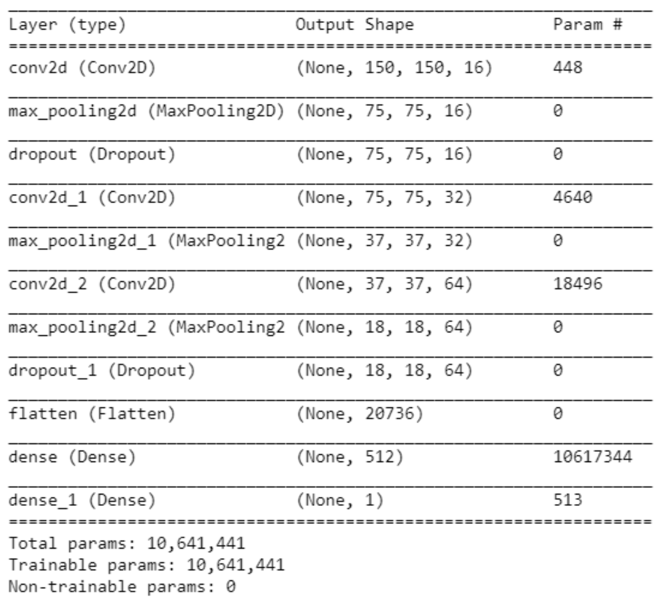
```
train_data_gen =
train_image_generator.flow_from_directory(batch_size=batch_size,
directory=TRAIN_PATH,
shuffle=True,
target_size=(IMG_HEIGHT, IMG_WIDTH),
class_mode='binary')

val_data_gen =
validation_image_generator.flow_from_directory(batch_size=batch_size,
directory=VAL_PATH,
target_size=(IMG_HEIGHT, IMG_WIDTH),
class_mode='binary')
```



```
Found 20000 images belonging to 2 classes.
Found 5000 images belonging to 2 classes.
```

Create the Deep Convolutional Neural network model

```
#Build the model
model = Sequential([
    Conv2D(16, 3, padding='same', activation='relu',
           input_shape=(IMG_HEIGHT, IMG_WIDTH ,3)),
    MaxPooling2D(),
    Dropout(0.2),
    Conv2D(32, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Conv2D(64, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Dropout(0.2),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(1)
])

# Compile the model
model.compile(optimizer='adam',
loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
metrics=['accuracy'])

# print the model architecture
model.summary()
```



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 150, 150, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 75, 75, 16)        0
_________________________________________________________________
dropout (Dropout)            (None, 75, 75, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 75, 75, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 37, 37, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 37, 37, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 18, 18, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 18, 18, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 20736)             0
_________________________________________________________________
dense (Dense)                (None, 512)               10617344
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 10,641,441
Trainable params: 10,641,441
Non-trainable params: 0
```

Training the Model

We train the model for 50 epochs

```
history = model.fit_generator(
    train_data_gen,
    steps_per_epoch=1000,
    epochs=epochs,
    validation_data=val_data_gen,
    validation_steps=1000
)
```

Feature Visualization

Feature Visualization translates the internal features present in an image into visually perceptible or recognizable image patterns. Feature visualization will help us understand the learned features explicitly.

First, you will visualize the different filters or feature detectors that are applied to the input image and, in the next step, visualize the feature maps or activation maps that are generated.

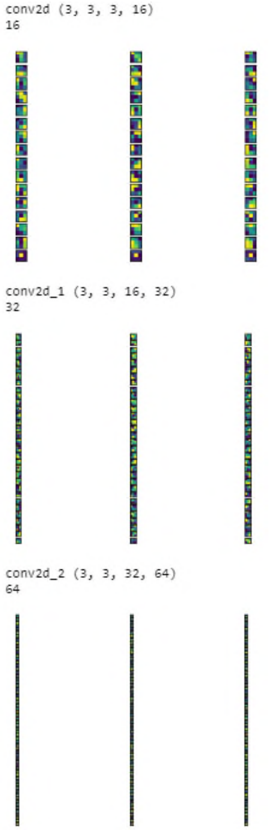
Visualizing Filters or Feature Detectors in a CNN

CNN uses learned filters to convolve the feature maps from the previous layer. Filters are two- dimensional weights and these weights have a spatial relationship with each other.

The steps you will follow to visualize the filters.

1. Iterate through all the layers of the model using ***model.layers***
2. If the layer is a convolutional layer, then extract the weights and bias values using *get_weights()* for that layer.
3. Normalize the weights for the filters between 0 and 1
4. Plot the filters for each of the convolutional layers and all the channels. For Color image, you will have three channels for RGB. For a grayscale image, the number of channels will be 1

```
#Iterate thru all the layers of the model
for layer in model.layers:
    if 'conv' in layer.name:
        weights, bias= layer.get_weights()
        print(layer.name, filters.shape)

        #normalize filter values between  0 and 1 for visualization
        f_min, f_max = weights.min(), weights.max()
        filters = (weights - f_min) / (f_max - f_min)
        print(filters.shape[3])
        filter_cnt=1

        #plotting all the filters
        for i in range(filters.shape[3]):
            #get the filters
            filt=filters[:,:,:, i]
            #plotting each of the channel, color image RGB channels
            for j in range(filters.shape[0]):
                ax= plt.subplot(filters.shape[3], filters.shape[0],
filter_cnt  )
                ax.set_xticks([])
                ax.set_yticks([])
                plt.imshow(filt[:,:, j])
                filter_cnt+=1
        plt.show()
```



```
conv2d (3, 3, 3, 16)
16
conv2d_1 (3, 3, 16, 32)
32
conv2d_2 (3, 3, 32, 64)
64
```

Filters applied to the CNN model for cats and dogs.

Visualizing Feature maps or Activation maps generated in a CNN

Feature maps are generated by applying Filters or Feature detectors to the input image or the feature map output of the prior layers. Feature map visualization will provide insight into the internal representations for specific input for each of the Convolutional layers in the model.

The steps you will follow to visualize the feature maps.

1. Define a new model, ***visualization_model*** that will take an image as the input. The output of the model will be feature maps, which are an intermediate representation for all layers after the first layer. This is based on the model we have used for training.
2. Load the input image for which we want to view the Feature map to understand which features were prominent to classify the image.

3. Convert the image to NumPy array

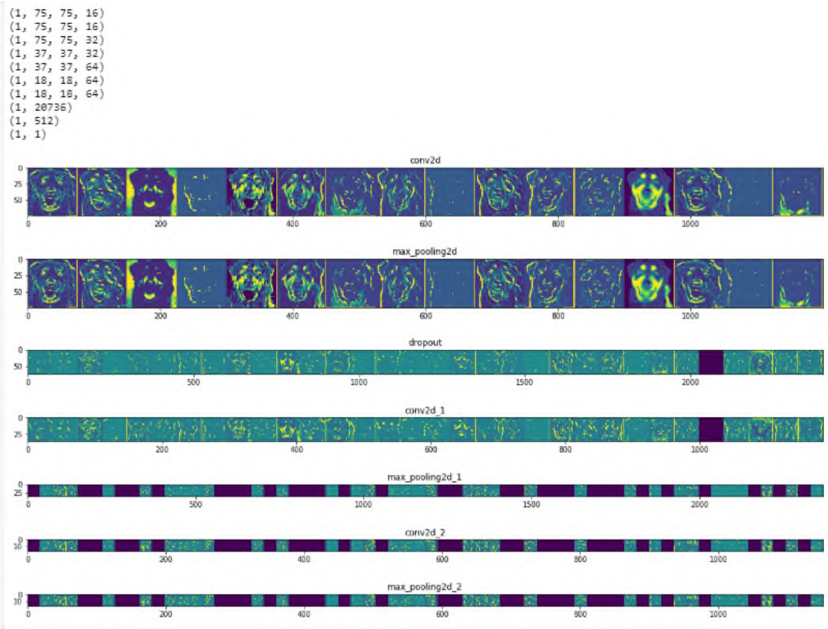
4. Normalize the array by rescaling it

5. Run the input image through the visualization model to obtain all intermediate representations for the input image.

6. Create the plot for all of the convolutional layers and the max pool layers but not for the fully connected layer. For plotting the Feature maps, retrieve the layer name for each of the layers in the model.

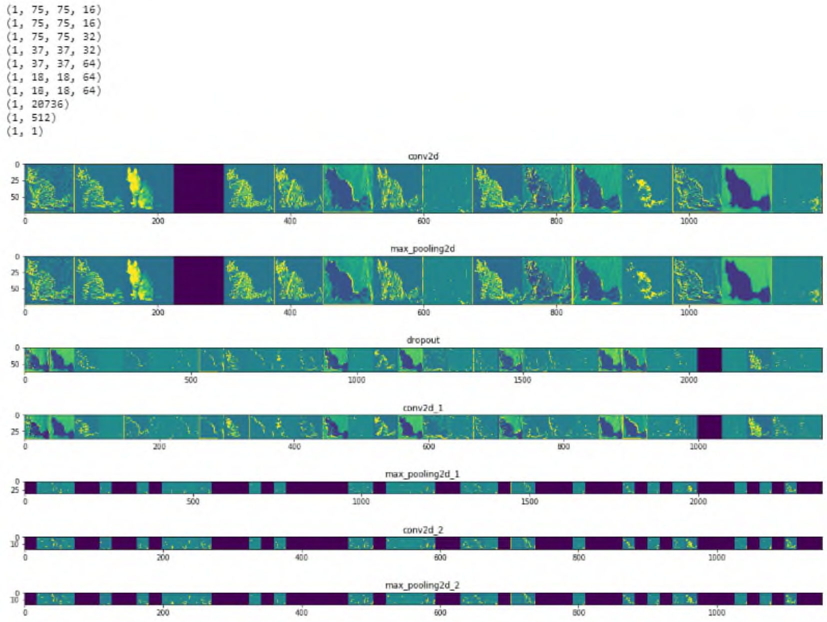
```
img_path='\\dogs-vs-cats\\test1\\137.jpg' #dog
# Define a new Model, Input= image
# Output= intermediate representations for all layers in the
# previous model after the first.
successive_outputs = [layer.output for layer in model.layers[1:]]

#visualization_model = Model(img_input, successive_outputs)
visualization_model = tf.keras.models.Model(inputs = model.input,
outputs = successive_outputs)

#Load the input image
img = load_img(img_path, target_size=(150, 150))

# Convert ht image to Array of dimension (150,150,3)
x   = img_to_array(img)
x   = x.reshape((1,) + x.shape)

# Rescale by 1/255
x /= 255.0

# Let's run input image through our vislauization network
# to obtain all intermediate representations for the image.
successive_feature_maps = visualization_model.predict(x)

# Retrieve are the names of the layers, so can have them as part of
our plot
layer_names = [layer.name for layer in model.layers]
for layer_name, feature_map in zip(layer_names,
successive_feature_maps):
  print(feature_map.shape)
  if len(feature_map.shape) == 4:

    # Plot Feature maps for the conv / maxpool layers, not the fully-
connected layers

    n_features = feature_map.shape[-1]  # number of features in the
feature map
    size       = feature_map.shape[ 1]  # feature map shape (1, size,
size, n_features)

    # We will tile our images in this matrix
    display_grid = np.zeros((size, size * n_features))

    # Postprocess the feature to be visually palatable
    for i in range(n_features):
      x  = feature_map[0, :, :, i]
      x -= x.mean()
      x /= x.std ()
      x *=  64
      x += 128
      x  = np.clip(x, 0, 255).astype('uint8')
```

```
   # Tile each filter into a horizontal grid
   display_grid[:, i * size : (i + 1) * size] = x

# Display the grid
  scale = 20. / n_features
  plt.figure( figsize=(scale * n_features, scale) )
  plt.title ( layer_name )
  plt.grid  ( False )
  plt.imshow( display_grid, aspect='auto', cmap='viridis' )
```



```
(1, 75, 75, 16)
(1, 75, 75, 16)
(1, 75, 75, 32)
(1, 37, 37, 32)
(1, 37, 37, 64)
(1, 18, 18, 64)
(1, 18, 18, 64)
(1, 20736)
(1, 512)
(1, 1)
```

Feature map for a dog image



```
(1, 75, 75, 16)
(1, 75, 75, 16)
(1, 75, 75, 32)
(1, 37, 37, 32)
(1, 37, 37, 64)
(1, 18, 18, 64)
(1, 18, 18, 64)
(1, 20736)
(1, 512)
(1, 1)
```

Feature map for a cat image

We can see that for the dog image, the snout and the tongue are very prominent features, and for the cat image, the ears and tail are prominent in the feature maps.

Code available here

Conclusion:

Visualizing an inside story of how CNN learns to identify different features present in images provides a deeper insight into how the model works. It will also help to understand why the model might be failing to classify some of the images correctly and hence fine-tuning the model for better accuracy and precision.

References:

Keras documentation: Keras layers API

Layers are the basic building blocks of neural networks in Keras. A layer consists of a tensor-in tensor-out...

keras.io

Get started Sign In

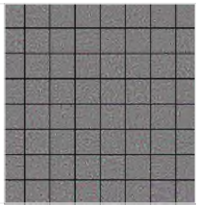
Search

Renu Khandelwal

Convolution filter visualization — Keras Documentation

from __future__ import print_function import time import numpy as np from PIL import Image as pil_image from...

keras.io



[How Convolutional Neural Networks see the World — A survey of Convolutional Neural Network Visualization Methods]

[Visualizing and Understanding Convolutional Networks]

[Feature Visualization — Google AI Blog]

https://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. Take a look.

By signing up, you will create a Medium account if you don't already have one. Review our Privacy Policy for more information about our privacy practices.

Get this newsletter

4.93K Followers

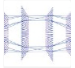
Loves learning, sharing, and discovering myself. Passionate about Machine Learning and Deep Learning

Follow

More from Medium

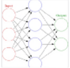
Dharmaraj

Convolutional Neural Networks (CNN) —...



Roopa CM

Activation Functions in Artificial Neural...

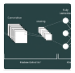


Ruks... in Towar...

Convolutional Neural Network (CNN)...

Ali... in students...

Training a Convolutional Neural Network...



Help Status Writers Blog Careers Privacy Terms About Text to speech

EXHIBIT J

Machine Learning Mastery
Making Developers Awesome at Machine Learning

How Do Convolutional Layers Work in Deep Learning Neural Networks?

by Jason Brownlee on April 17, 2019 in Deep Learning for Computer Vision

Last Updated on April 17, 2020

Convolutional layers are the major building blocks used in convolutional neural networks.

A convolution is the simple application of a filter to an input that results in an activation. Repeated application of the same filter to an input results in a map of activations called a feature map, indicating the locations and strength of a detected feature in an input, such as an image.

The innovation of convolutional neural networks is the ability to automatically learn a large number of filters in parallel specific to a training dataset under the constraints of a specific predictive modeling problem, such as image classification. The result is highly specific features that can be detected anywhere on input images.

In this tutorial, you will discover how convolutions work in the convolutional neural network.

After completing this tutorial, you will know:

- Convolutional neural networks apply a filter to an input to create a feature map that summarizes the presence of detected features in the input.
- Filters can be handcrafted, such as line detectors, but the innovation of convolutional neural networks is to learn the filters during training in the context of a specific prediction problem.
- How to calculate the feature map for one- and two-dimensional convolutional layers in a convolutional neural network.

Kick-start your project with my new book Deep Learning for Computer Vision, including *step-by-step tutorials* and the *Python source code* files for all examples.

Let's get started.

A Gentle Introduction to Convolutional Layers for Deep Learning Neural Networks
Photo by mendhak, some rights reserved.

Tutorial Overview

This tutorial is divided into four parts; they are:

1. Convolution in Convolutional Neural Networks
2. Convolution in Computer Vision
3. Power of Learned Filters
4. Worked Example of Convolutional Layers

Convolution in Convolutional Neural Networks

The convolutional neural network, or CNN for short, is a specialized type of neural network model designed for working with two-dimensional image data, although they can be used with one-dimensional and three-dimensional data.

Central to the convolutional neural network is the convolutional layer that gives the network its name. This layer performs an operation called a "*convolution*".

In the context of a convolutional neural network, a convolution is a linear operation that involves the multiplication of a set of weights with the input, much like a traditional neural network. Given that the technique was designed for two-dimensional input, the multiplication is performed between an array of input data and a two-dimensional array of weights, called a filter or a kernel.

The filter is smaller than the input data and the type of multiplication applied between a filter-sized patch of the input and the filter is a dot product. A dot product is the element-wise multiplication between the filter-sized patch of the input and filter, which is then summed, always resulting in a single value. Because it results in a single value, the operation is often referred to as the "*scalar product*".

Using a filter smaller than the input is intentional as it allows the same filter (set of weights) to be multiplied by the input array multiple times at different points on the input. Specifically, the filter is applied systematically to each overlapping part or filter-sized patch of the input data, left to right, top to bottom.

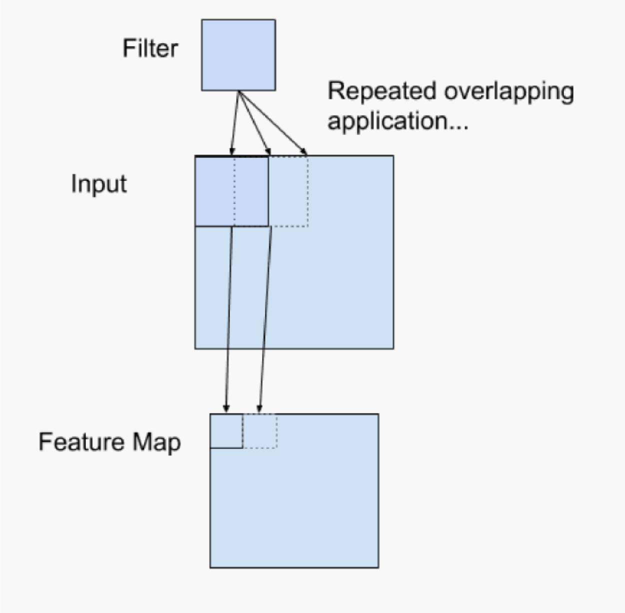
This systematic application of the same filter across an image is a powerful idea. If the filter is designed to detect a specific type of feature in the input, then the application of that filter systematically across the entire input image allows the filter an opportunity to discover that feature anywhere in the image. This capability is commonly referred to as translation invariance, e.g. the general interest in whether the feature is present rather than where it was present.

> *Invariance to local translation can be a very useful property if we care more about whether some feature is present than exactly where it is. For example, when determining whether an image contains a face, we need not know the location of the eyes with pixel-perfect accuracy, we just need to know that there is an eye on the left side of the face and an eye on the right side of the face.*

— Page 342, Deep Learning, 2016.

The output from multiplying the filter with the input array one time is a single value. As the filter is applied multiple times to the input array, the result is a two-dimensional array of output values that represent a filtering of the input. As such, the two-dimensional output array from this operation is called a "*feature map*".

Once a feature map is created, we can pass each value in the feature map through a nonlinearity, such as a ReLU, much like we do for the outputs of a fully connected layer.



Example of a Filter Applied to a Two-Dimensional Input to Create a Feature Map

If you come from a digital signal processing field or related area of mathematics, you may understand the convolution operation on a matrix as something different. Specifically, the filter (kernel) is flipped prior to being applied to the input. Technically, the convolution as described in the use of convolutional neural networks is actually a "*cross-correlation*". Nevertheless, in deep learning, it is referred to as a "*convolution*" operation.

> *Many machine learning libraries implement cross-correlation but call it convolution.*

— Page 333, Deep Learning, 2016.

In summary, we have a *input*, such as an image of pixel values, and we have a *filter*, which is a set of weights, and the filter is systematically applied to the input data to create a *feature map*.

Convolution in Computer Vision

The idea of applying the convolutional operation to image data is not new or unique to convolutional neural networks; it is a common technique used in computer vision.

Historically, filters were designed by hand by computer vision experts, which were then applied to an image to result in a feature map or output from applying the filter then makes the analysis of the image easier in some way.

For example, below is a hand crafted 3×3 element filter for detecting vertical lines:

```
0.0, 1.0, 0.0
0.0, 1.0, 0.0
0.0, 1.0, 0.0
```

Applying this filter to an image will result in a feature map that only contains vertical lines. It is a vertical line detector.

You can see this from the weight values in the filter; any pixels values in the center vertical line will be positively activated and any on either side will be negatively activated. Dragging this filter systematically across pixel values in an image can only highlight vertical line pixels.

A horizontal line detector could also be created and also applied to the image, for example:

```
0.0, 0.0, 0.0
1.0, 1.0, 1.0
0.0, 0.0, 0.0
```

Combining the results from both filters, e.g. combining both feature maps, will result in all of the lines in an image being highlighted.

A suite of tens or even hundreds of other small filters can be designed to detect other features in the image.

The innovation of using the convolution operation in a neural network is that the values of the filter are weights to be learned during the training of the network.

The network will learn what types of features to extract from the input. Specifically, training under stochastic gradient descent, the network is forced to learn to extract features from the image that minimize the loss for the specific task the network is being trained to solve, e.g. extract features that are the most useful for classifying images as dogs or cats.

In this context, you can see that this is a powerful idea.

Power of Learned Filters

Learning a single filter specific to a machine learning task is a powerful technique.

Yet, convolutional neural networks achieve much more in practice.

Multiple Filters

Convolutional neural networks do not learn a single filter; they, in fact, learn multiple features in parallel for a given input.

For example, it is common for a convolutional layer to learn from 32 to 512 filters in parallel for a given input.

This gives the model 32, or even 512, different ways of extracting features from an input, or many different ways of both "*learning to see*" and after training, many different ways of "*seeing*" the input data.

This diversity allows specialization, e.g. not just lines, but the specific lines seen in your specific training data.

Multiple Channels

Color images have multiple channels, typically one for each color channel, such as red, green, and blue.

From a data perspective, that means that a single image provided as input to the model is, in fact, three images.

A filter must always have the same number of channels as the input, often referred to as "*depth*". If an input image has 3 channels (e.g. a depth of 3), then a filter applied to that image must also have 3 channels (e.g. a depth of 3). In this case, a 3×3 filter would in fact be 3x3x3 or [3, 3, 3] for rows, columns, and depth. Regardless of the depth of the input and depth of the filter, the filter is applied to the input using a dot product operation which results in a single value.

This means that if a convolutional layer has 32 filters, these 32 filters are not just two-dimensional for the two-dimensional image input, but are also three-dimensional, having specific filter weights for each of the three channels. Yet, each filter results in a single feature map. Which means that the depth of the output of applying the convolutional layer with 32 filters is 32 for the 32 feature maps created.

Multiple Layers

Convolutional layers are not only applied to input data, e.g. raw pixel values, but they can also be applied to the output of other layers.

The stacking of convolutional layers allows a hierarchical decomposition of the input.

Consider that the filters that operate directly on the raw pixel values will learn to extract low-level features, such as lines.

The filters that operate on the output of the first line layers may extract features that are combinations of lower-level features, such as features that comprise multiple lines to express shapes.

This process continues until very deep layers are extracting faces, animals, houses, and so on.

This is exactly what we see in practice. The abstraction of features to high and higher orders as the depth of the network is increased.

Worked Example of Convolutional Layers

The Keras deep learning library provides a suite of convolutional layers.

We can better understand the convolution operation by looking at some worked examples with contrived data and handcrafted filters.

In this section, we'll look at both a one-dimensional convolutional layer and a two-dimensional convolutional layer example to both make the convolution operation concrete and provide a worked example of using the Keras layers.

Example of 1D Convolutional Layer

We can define a one-dimensional input that has eight elements all with the value of 0.0, with a two element bump in the middle with the values 1.0.

```
[0, 0, 0, 1, 1, 0, 0, 0]
```

The input to Keras must be three dimensional for a 1D convolutional layer.

The first dimension refers to each input sample; in this case, we only have one sample. The second dimension refers to the length of each sample; in this case, the length is eight. The third dimension refers to the number of channels in each sample; in this case, we only have a single channel.

Therefore, the shape of the input array will be [1, 8, 1].

```
# define input data
data = asarray([0, 0, 0, 1, 1, 0, 0, 0])
data = data.reshape(1, 8, 1)
```

We will define a model that expects input samples to have the shape [8, 1].

The model will have a single filter with the shape of 3, or three elements wide. Keras refers to the shape of the filter as the *kernel_size*.

```
# create model
model = Sequential()
model.add(Conv1D(1, 3, input_shape=(8, 1)))
```

By default, the filters in a convolutional layer are initialized with random weights. In this contrived example, we will manually specify the weights for the single filter. We will define a filter that is capable of detecting bumps, that is a high input value surrounded by low input values, as we defined in our input example.

The three element filter we will define looks as follows:

```
[0, 1, 0]
```

The convolutional layer also has a bias input value that also requires a weight that we will set to zero.

Therefore, we can force the weights of our one-dimensional convolutional layer to use our handcrafted filter as follows:

```
# define a vertical line detector
weights = [asarray([[[0]],[[1]],[[0]]]), asarray([0.0])]
# store the weights in the model
model.set_weights(weights)
```

The weights must be specified in a three-dimensional structure, in terms of rows, columns, and channels. The filter has a single row, three columns, and one channel.

We can retrieve the weights and confirm that they were set correctly.

```
# confirm they were stored
print(model.get_weights())
```

Finally, we can apply the single filter to our input data.

We can achieve this by calling the *predict()* function on the model. This will return the feature map directly: that is the output of applying the filter systematically across the input sequence.

```
# apply filter to input data
yhat = model.predict(data)
print(yhat)
```

Tying all of this together, the complete example is listed below.

```
# example of calculation 1d convolutions
from numpy import asarray
from keras.models import Sequential
from keras.layers import Conv1D
# define input data
data = asarray([0, 0, 0, 1, 1, 0, 0, 0])
data = data.reshape(1, 8, 1)
# create model
model = Sequential()
model.add(Conv1D(1, 3, input_shape=(8, 1)))
# define a vertical line detector
weights = [asarray([[[0]],[[1]],[[0]]]), asarray([0.0])]
# store the weights in the model
model.set_weights(weights)
# confirm they were stored
print(model.get_weights())
# apply filter to input data
yhat = model.predict(data)
print(yhat)
```

Running the example first prints the weights of the network; that is the confirmation that our handcrafted filter was set in the model as we expected.

Next, the filter is applied to the input pattern and the feature map is calculated and displayed. We can see from the values of the feature map that the bump was detected correctly.

```
[array([[[0.]],
       [[1.]],
       [[0.]]], dtype=float32), array([0.], dtype=float32)]

[[[0.]
  [0.]
  [1.]
  [1.]
  [0.]
  [0.]]]
```

Let's take a closer look at what happened here.

Recall that the input is an eight element vector with the values: [0, 0, 0, 1, 1, 0, 0, 0].

First, the three-element filter [0, 1, 0] was applied to the first three inputs of the input [0, 0, 0] by calculating the dot product ("." operator), which resulted in a single output value in the feature map of zero.

Recall that a dot product is the sum of the element-wise multiplications, or here it is (0 x 0) + (1 x 0) + (0 x 0) = 0. In NumPy, this can be implemented manually as:

```
from numpy import asarray
print(asarray([0, 1, 0]).dot(asarray([0, 0, 0])))
```

In our manual example, this is as follows:

```
[0, 1, 0] . [0, 0, 0] = 0
```

The filter was then moved along one element of the input sequence and the process was repeated; specifically, the same filter was applied to the input sequence at indexes 1, 2, and 3, which also resulted in a zero output in the feature map.

```
[0, 1, 0] . [0, 0, 1] = 0
```

We are being systematic, so again, the filter is moved along one more element of the input and applied to the input at indexes 2, 3, and 4. This time the output is a value of one in the feature map. We detected the feature and activated appropriately.

```
[0, 1, 0] . [0, 1, 1] = 1
```

The process is repeated until we calculate the entire feature map.

```
[0, 0, 1, 1, 0, 0]
```

Note that the feature map has six elements, whereas our input has eight elements. This is an artefact of how the filter was applied to the input sequence. There are other ways to apply the filter to the input sequence that changes the shape of the resulting feature map, such as padding, but we will not discuss these methods in this post.

You can imagine that with different inputs, we may detect the feature with more or less intensity, and with different weights in the filter, that we would detect different features in the input sequence.

Example of 2D Convolutional Layer

We can expand the bump detection example in the previous section to a vertical line detector in a two-dimensional image.

Again, we can constrain the input, in this case to a square 8×8 pixel input image with a single channel (e.g. grayscale) with a single vertical line in the middle.

```
[0, 0, 0, 1, 1, 0, 0, 0]
[0, 0, 0, 1, 1, 0, 0, 0]
[0, 0, 0, 1, 1, 0, 0, 0]
[0, 0, 0, 1, 1, 0, 0, 0]
[0, 0, 0, 1, 1, 0, 0, 0]
[0, 0, 0, 1, 1, 0, 0, 0]
[0, 0, 0, 1, 1, 0, 0, 0]
[0, 0, 0, 1, 1, 0, 0, 0]
```

The input to a Conv2D layer must be four-dimensional.

The first dimension defines the samples; in this case, there is only a single sample. The second dimension defines the number of rows; in this case, eight. The third dimension defines the number of columns, again eight in this case, and finally the number of channels, which is one in this case.

Therefore, the input must have the four-dimensional shape [samples, rows, columns, channels] or [1, 8, 8, 1] in this case.

```
# define input data
data = [[0, 0, 0, 1, 1, 0, 0, 0],
        [0, 0, 0, 1, 1, 0, 0, 0],
        [0, 0, 0, 1, 1, 0, 0, 0],
        [0, 0, 0, 1, 1, 0, 0, 0],
        [0, 0, 0, 1, 1, 0, 0, 0],
        [0, 0, 0, 1, 1, 0, 0, 0],
        [0, 0, 0, 1, 1, 0, 0, 0],
        [0, 0, 0, 1, 1, 0, 0, 0]]
data = asarray(data)
data = data.reshape(1, 8, 8, 1)
```

We will define the Conv2D with a single filter as we did in the previous section with the Conv1D example.

The filter will be two-dimensional and square with the shape 3×3. The layer will expect input samples to have the shape [columns, rows, channels] or [8,8,1].

```
# create model
model = Sequential()
model.add(Conv2D(1, (3,3), input_shape=(8, 8, 1)))
```

We will define a vertical line detector filter to detect the single vertical line in our input data.

The filter looks as follows:

```
0, 1, 0
0, 1, 0
0, 1, 0
```

We can implement this as follows:

```
# define a vertical line detector
detector = [[[[0]],[[1]],[[0]]],
            [[[0]],[[1]],[[0]]],
            [[[0]],[[1]],[[0]]]]
```

```
weights = [asarray(detector), asarray([0.0])]
# store the weights in the model
model.set_weights(weights)
# confirm they were stored
print(model.get_weights())
```

Finally, we will apply the filter to the input image, which will result in a feature map that we would expect to show the detection of the vertical line in the input image.

```
# apply filter to input data
yhat = model.predict(data)
```

The shape of the feature map output will be four-dimensional with the shape [batch, rows, columns, filters]. We will be performing a single batch and we have a single filter (one filter and one input channel), therefore the output shape is [1, ?, ?, 1]. We can pretty-print the content of the single feature map as follows:

```
for r in range(yhat.shape[1]):
    # print each column in the row
    print([yhat[0,r,c,0] for c in range(yhat.shape[2])])
```

Tying all of this together, the complete example is listed below.

```
# example of calculation 2d convolutions
from numpy import asarray
from keras.models import Sequential
from keras.layers import Conv2D
# define input data
data = [[0, 0, 0, 1, 1, 0, 0, 0],
        [0, 0, 0, 1, 1, 0, 0, 0],
        [0, 0, 0, 1, 1, 0, 0, 0],
        [0, 0, 0, 1, 1, 0, 0, 0],
        [0, 0, 0, 1, 1, 0, 0, 0],
        [0, 0, 0, 1, 1, 0, 0, 0],
        [0, 0, 0, 1, 1, 0, 0, 0],
        [0, 0, 0, 1, 1, 0, 0, 0]]
data = asarray(data)
data = data.reshape(1, 8, 8, 1)
# create model
model = Sequential()
model.add(Conv2D(1, (3,3), input_shape=(8, 8, 1)))
# define a vertical line detector
detector = [[[[0]],[[1]],[[0]]],
            [[[0]],[[1]],[[0]]],
            [[[0]],[[1]],[[0]]]]
weights = [asarray(detector), asarray([0.0])]
# store the weights in the model
model.set_weights(weights)
# confirm they were stored
print(model.get_weights())
# apply filter to input data
yhat = model.predict(data)
for r in range(yhat.shape[1]):
    # print each column in the row
    print([yhat[0,r,c,0] for c in range(yhat.shape[2])])
```

Running the example first confirms that the handcrafted filter was correctly defined in the layer weights

Next, the calculated feature map is printed. We can see from the scale of the numbers that indeed the filter has detected the single vertical line with strong activation in the middle of the feature map.

```
[array([[[[0.]],
        [[1.]],
        [[0.]]],
       [[[0.]],
        [[1.]],
        [[0.]]],
       [[[0.]],
        [[1.]],
       [[0.]]]], dtype=float32), array([0.], dtype=float32)]

[0.0, 0.0, 3.0, 3.0, 0.0, 0.0]
[0.0, 0.0, 3.0, 3.0, 0.0, 0.0]
[0.0, 0.0, 3.0, 3.0, 0.0, 0.0]
[0.0, 0.0, 3.0, 3.0, 0.0, 0.0]
[0.0, 0.0, 3.0, 3.0, 0.0, 0.0]
[0.0, 0.0, 3.0, 3.0, 0.0, 0.0]
```

Let’s take a closer look at what was calculated.

First, the filter was applied to the top left corner of the image, or an image patch of 3×3 elements. Technically, the image patch is three dimensional with a single channel, and the filter has the same dimensions. We cannot implement this in NumPy using the dot() function, instead, we must use the tensordot() function so we can appropriately sum across all dimensions, for example:

```
from numpy import asarray
from numpy import tensordot
m1 = asarray([[0, 1, 0],
              [0, 1, 0],
              [0, 1, 0]])
m2 = asarray([[0, 0, 0],
              [0, 0, 0],
              [0, 0, 0]])
print(tensordot(m1, m2))
```

This calculation results in a single output value of 0.0, e.g., the feature was not detected. This gives us the first element in the top-left corner of the feature map.

Manually, this would be as follows:

```
0, 1, 0     0, 0, 0
0, 1, 0  .  0, 0, 0 = 0
0, 1, 0     0, 0, 0
```

The filter is moved along one column to the left and the process is repeated. Again, the feature is not detected.

```
0, 1, 0     0, 0, 1
0, 1, 0  .  0, 0, 1 = 0
0, 1, 0     0, 0, 1
```

One more move to the left to the next column and the feature is detected for the first time, resulting in a strong activation.

```
0, 1, 0     0, 1, 1
0, 1, 0  .  0, 1, 1 = 3
0, 1, 0     0, 1, 1
```

This process is repeated until the edge of the filter rests against the edge or final column of the input image. This gives the last element in the first full row of the feature map.

```
[0.0, 0.0, 3.0, 3.0, 0.0, 0.0]
```

The filter then moves down one row and back to the first column and the process is related from left to right to give the second row of the feature map. And on until the bottom of the filter rests on the bottom or last row of the input image.

Again, as with the previous section, we can see that the feature map is a 6×6 matrix, smaller than the 8×8 input image because of the limitations of how the filter can be applied to the input image.

Further Reading

This section provides more resources on the topic if you are looking to go deeper.

Posts

- Crash Course in Convolutional Neural Networks for Machine Learning

Books

- Chapter 9: Convolutional Networks, Deep Learning, 2016.
- Chapter 5: Deep Learning for Computer Vision, Deep Learning with Python, 2017.

API

- Keras Convolutional Layers API
- numpy.asarray API

Summary

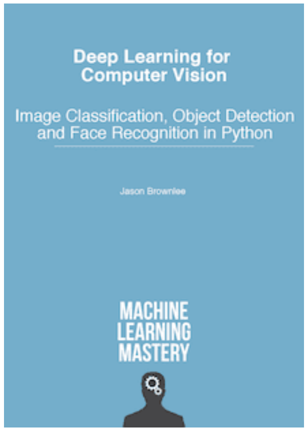
In this tutorial, you discovered how convolutions work in the convolutional neural network.

Specifically, you learned:

- Convolutional neural networks apply a filter to an input to create a feature map that summarizes the presence of detected features in the input.
- Filters can be handcrafted, such as line detectors, but the innovation of convolutional neural networks is to learn the filters during training in the context of a specific prediction problem.
- How to calculate the feature map for one- and two-dimensional convolutional layers in a convolutional neural network.

Do you have any questions?
Ask your questions in the comments below and I will do my best to answer.

Develop Deep Learning Models for Vision Today!



Deep Learning for Computer Vision

Image Classification, Object Detection and Face Recognition in Python

Jason Brownlee

MACHINE LEARNING MASTERY

Develop Your Own Vision Models in Minutes

...with just a few lines of python code

Discover how in my new Ebook:
Deep Learning for Computer Vision

It provides **self-study tutorials** on topics like:
classification, *object detection (yolo and rcnn)*, *face recognition (vggface and facenet)*, *data preparation* and much more...

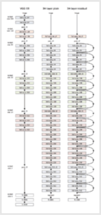
Finally Bring Deep Learning to your Vision Projects

Skip the Academics. Just Results.

SEE WHAT'S INSIDE

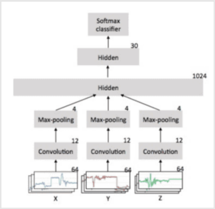
Tweet Tweet Share Share

More On This Topic



Convolutional Neural Network Model Innovations for…

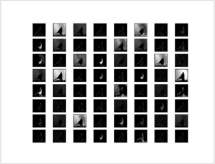
How to Develop VGG, Inception and ResNet Modules…



Deep Learning Models for Human Activity Recognition

How to use the UpSampling2D and Conv2DTranspose…

Crash Course in Convolutional Neural Networks for…



How to Visualize Filters and Feature Maps in…

About Jason Brownlee

Jason Brownlee, PhD is a machine learning specialist who teaches developers how to get results with modern machine learning methods via hands-on tutorials.
View all posts by Jason Brownlee →

‹ How to Use Test-Time Augmentation to Make Better Predictions

A Gentle Introduction to Padding and Stride for Convolutional Neural Networks ›

76 Responses to *How Do Convolutional Layers Work in Deep Learning Neural Networks?*

SHAHEEN ALHIRMIZY April 19, 2019 at 4:36 pm #

REPLY ↰

THANK you very much for your excellent explanations, I have two questions :
first one about how to fine tuning filters in convolution in order to extract specific feature from input images I mean can we change the filter values and how?.

second question why 32 to 512 filters? and are the values of these filters assumed by the model in stochastic way?

Jason Brownlee April 20, 2019 at 7:31 am #

REPLY ↰

Thanks. Great questions!

No, the filter values (weights) are learned. They are typically not manually modified/specified.

The number of filters is a hyperparameter that is best set via trial and error:
https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/

Isha October 16, 2022 at 5:00 pm #

REPLY ↰

Same Question, I wanted to ask.
How can we choose filter's weight if we can not then, do we take random values?
these filter weights can be learned during the training process? How?

James Carmichael October 17, 2022 at 10:38 am #

REPLY ↰

Hi Isha…The following resource may add clarity:

https://machinelearningmastery.com/why-initialize-a-neural-network-with-random-weights/

prisilla April 21, 2019 at 12:46 am #

REPLY ↰

Good Explanation!

My query is
Why do the parameters in pooling and flatten equal to zero? Is it only because while pooling -maxpooling or average pooling, the number of nodes are reduced.
And for flatten as it is converted to a single dimension array.

Jason Brownlee April 21, 2019 at 8:24 am #

REPLY

Great question!

Those layers have no weights, they just transform the shape of the input in the case of flatten, or select a subset of values in the case of poling.

prisilla April 22, 2019 at 1:33 am #

REPLY

Thanks Jason

James April 28, 2019 at 2:54 pm #

REPLY

Thanks Jason for great article!
Could you clarify a couple of things for me?
First, is number of filters equals to number of feature maps?
It makes sense to me that layers closer to the input layer detect features like lines and shapes, and layers closer to the output detect more concret objects like desk and chairs.
However, don't we need more number of filters to detect many small shapes and lines in the beginning of the network close to the input, and narrow them down as it gets closer to the output?
Also I would like to think that it's better to start with smaller window (kernel) size close to the input and makes it bigger toward the output.
This makes sense in my head, but obviously this is not correct.
In models I've seen so far, number of filters increases, and the window size seems to stays static.
Would you mind explaining how it works?

Jason Brownlee April 29, 2019 at 8:15 am #

REPLY

Yes, the number of filters == the number of feature maps, in general.

Intuitions between the number fo filters and filter sizes and what they are detecting seem to breakdown. Model architectures are empirical, not based on theory, for example:
https://machinelearningmastery.com/review-of-architectural-innovations-for-convolutional-neural-networks-for-image-classification/

Rasmus Hartvig May 19, 2019 at 8:19 pm #

REPLY

First, thanks a million for some wonderful articles, very well presented!

I wondered, if you stack convolutional layers, each with > 1 filter, it seems the number of dimensions would be increasing. E.g. for a 2D image, first conv layer produces a 2D x number of filters, ie 3D. This becomes the input to second layer, which in turn produces 3D x number of filters of second conv layer, ie 4D.

From searching around*, I understand one may avoid this by making the third dimension in second layer equal to number of filters of first layer. Thus the second layer still produces only 3 dimensions.

Can you comment on this approach? If incorrect or subtleties are overlooked, maybe it's worth adding a section on sequential convolutional layers to the article.

* https://datascience.stackexchange.com/questions/9175/how-do-subsequent-convolution-layers-work?newreg=82cdb799f5f04512a8c00e2a7b445c95

Thanks again!

Jason Brownlee May 20, 2019 at 6:27 am #

REPLY

Good question.

The size of the filter will shrink the input area. "same" padding can be used to avoid this.

The number of filters defines the channel or third dimension output. This does not linearly increase as one filter apply down through all channels in the input. Therefore at each layer you can choose the output depth/channels as the number of filters.

Does that help?

Noushin June 24, 2019 at 7:50 pm #

REPLY

First of all, thanks a lot for all the tutorials. Well presented tutorials about basic and essential information saved me many times. I've been using CNN for a while and as far as I search and study, one question still remained without an answer. Based on my understanding each conv layer extracts specific types of features. Let's say the first layer extracts all sorts of edge features (i.e horizontal, vertical, diagonal, etc.), As a result, the output of the layer are many images each showing some sort of edges. The second layer is supposed to extract texture features. Since the output of the first layer is not the original image anymore, how does the second layer extract textures out of it? Maybe my question is absurd or I did not understand the aim of convolution operation correctly. Yet, I appreciate if you correct me. Once again, thanks a lot for your tutorials and demonstrated codes.

Jason Brownlee June 25, 2019 at 6:18 am #

REPLY

Yes, the layers close to input extract simple features and the layers closer to output extract higher order features.

Each filter is different, so we are extracting 128 or 256 different features at a given layer.

This might help to give you an example of what is being extracted:
https://machinelearningmastery.com/how-to-visualize-filters-and-feature-maps-in-convolutional-neural-networks/

Sung April 16, 2020 at 7:42 pm #

REPLY

Hi, thanks for your great article. The following paragraphs in the article puzzled me. Would you be so kind to shed light?

Extracted from the article:
"The first dimension defines the samples; in this case, there is only a single sample. The second dimension defines the number of rows; in this case, eight. The third dimension defines the number of columns, again eight in this case, and finally the number of channels, which is one in this case.

Therefore, the input must have the four-dimensional shape [samples, columns, rows, channels] or [1, 8, 8, 1] in this case."

Is it [samples, rows, columns, channels] rather than [samples, columns, rows, channels] ?

Jason Brownlee April 17, 2020 at 6:18 am #

REPLY

Looks like a typo, fixed. Thanks!

noushin June 25, 2019 at 6:57 pm #

REPLY

Oh, thank you. I never crossed that tutorial page. It is really helpful.

Jason Brownlee June 26, 2019 at 6:37 am #

REPLY

No problem.

AMI January 27, 2020 at 7:13 am #

REPLY

I intend to know about various lightweight cnn(deep learning Networks) and references

How lightweight cnn are different from series and DAG cnn Networks

Are shufflenet, mobilenetv2 and squeezenet models are lightweight

Jason Brownlee January 27, 2020 at 7:40 am #

REPLY

What do you mean by “lightweight cnn”?

Vishnu February 12, 2020 at 5:58 am #

REPLY

Hi,
I am presently working on CNN for recognizing hand written characters belonging to a specific language using matlab.
After training the CNN , I am getting an validation set accuracy of about 90% but during the testing phase , I am not getting satisfactory results (the characters are getting classified wrongly).

How to get satisfactory results in both training and testing phases?
Will you pls help me regarding this issue.
Thanks in advance!

Jason Brownlee February 12, 2020 at 1:35 pm #

REPLY

That is a large topic, you can get started here:
https://machinelearningmastery.com/start-here/#better

ohood fadil April 21, 2020 at 10:06 pm #

REPLY

hi can you help me?
I master student in computer science and I wont your email

Jason Brownlee April 22, 2020 at 5:55 am #

REPLY ↰

You can contact me any time here:
https://machinelearningmastery.com/contact/

Yogeeshwari February 18, 2020 at 3:31 am #

REPLY ↰

Sir, How can I use conv2D layers as my classification output layer for 10 class classification instead of the dense layer?

Thanks in advance!

Jason Brownlee February 18, 2020 at 6:23 am #

REPLY ↰

Add a global pooling layer.

Hanan April 18, 2020 at 6:56 am #

REPLY ↰

Great Tutorial

Jason Brownlee April 18, 2020 at 1:41 pm #

REPLY ↰

Thanks!

Kasjan April 29, 2020 at 7:30 pm #

REPLY ↰

Hello Jason, you're website has been very helpful to me, thanks a lot!

I found an error here, in the beginning you write about translation invariance when referring
to the convolution filter being applied over the whole image. But you're quoting Goodfellow et. al. (p. 342) when they're talking about the Pooling Operation, not the Filter sliding over the whole image.

It should be **equivariance** to translation (p. 338) when we talk about the Filter sliding over
the image finding any features new position after a picture might be translation transformed.

Where translation invariance talks about the result of pooling being *exactly the same*, when the picture is translated by 1-2 Pixel (Because Pooling will return the same value).

Invariance: same result regardless of operation applied to prior: f(g(x)) = f(x)

equivariance: result changes accordingly to operation, i.e. the feature map output changes
when a feature appears somewhere else in the picture after translation.

p. 338: f(g(x)) = g(f(x))

Jason Brownlee April 30, 2020 at 6:41 am #

REPLY

Thanks, I'm happy to hear that.

Thanks!

Alok May 22, 2020 at 9:11 pm #

REPLY

Hi Jason. Thank you for the article. It is really insightful.
I have a doubt that is related to using two convolution layers stacked together versus a single convolution layer.
(a) For example, how is two convolution layers that are stacked together with say 8 filters for each layer, different from a single convolution layer with 8 filters? Is stacking two convolution layers help in identifying detailed features?
(b) For the case of two convolution layers stacked together, using different filters for each layer, like 8 for first and 16 for second, gives a better or worse learning that using same filters for both the layers?

Thanks in advance!

Jason Brownlee May 23, 2020 at 6:20 am #

REPLY

The next layer operates on the feature maps output by the first layer.

It can help on some problems.

Zi May 24, 2020 at 11:16 am #

REPLY

Hi, in the conv2D section, the article states "The filter is moved along one column to the left and the process is repeated. Again, the feature is not detected." But it looks as if the filter is moving to the right, since the 1's from the data are shifted in from the right.

Matthew August 20, 2020 at 4:14 am #

REPLY

Hey Jason I've been trying to find an article about the a 2d convolution but applied to an RGB image. In grayscale I understand, since it's just 1 channel. But when we have three channels the filter also has a depth of 3. I assume that the red layer matches up with a single layer of the filter and does a convolution much like the grayscale. However, aren't we left then we a feature map that has a depth of 3? One layer for each filter? I don't understand how the feature map comes out to a depth of 1 because it's one filter.

I understand that with multiple filters it is stacked, but how does one filter equate to one layer of depth?

Jason Brownlee August 20, 2020 at 6:53 am #

REPLY

Yes. One feature map per filter and channel.

This may help:
https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/

Maybe this will help:
https://machinelearningmastery.com/a-gentle-introduction-to-channels-first-and-channels-last-image-formats-for-deep-learning/

Kavita September 25, 2020 at 11:09 pm # REPLY

what happen if we decrease filter size In Cnn like 64,32,16 filters, instead of increasing filter size?

Jason Brownlee September 26, 2020 at 6:19 am # REPLY

It will change the capability and in turn the performance of the model. Try it and see.

Tom Cipollone October 23, 2020 at 6:05 am # REPLY

I am an older engineer that came out of the image processing industry, where we had to build our own convolution engines out of discrete multipliers and accumulators (we also built our own graphics cards and you were really "hot" if your PC ran at 6 MHz). Kernel filters for image processing were fixed as per application requirements.

My understanding of DNNs using CNNs is that the kernel filters are adjusted during the training process. I could be wrong but I'm not sure if the terminology for the kernel filters is now "weights". These "weights" are adjusted until a desired output of the DNN is reached.

I realize that there are many sets of weights representing the different convolutional filters that are used in the CNN stage. I also realize that to save space in memory this large number of weights is formatted.

My expectation is that each kernel filter would have to have its own unique space in system memory. My question is, is there a way to access the fully trained weights that act as the convolution filter? Yes, this would depend upon the formatting and the CNN framework that is used. I have done projects using the Darknet Framework and YOLO and I am currently learning Pytorch, but my question seems to be too basic.

Please correct any incorrect assumptions that I may have made.

Tom

Jason Brownlee October 23, 2020 at 6:20 am # REPLY

Yes, most APIs provide a way to extract the trained weights of the model. In keras it is model.get_weights() not sure about pytorch off the cuff.

You won't have one filter, you will have hundreds or thousands depending on the depth and complexity of the model.

Tom Cipollone October 24, 2020 at 4:06 am # REPLY

Thank you so much for your reply. Yes, of course, you are correct about the possible number of filters being in the hundreds or thousands.

Just one more question, that I hope is not too naive. Would it be true to say that there is a direct correlation, in terms of the number of filters in a CNN based DNN, and the work that the network is required to do? For example, if the network was trained to distinguish between 100 different object types as opposed to a single object type, would there be many more filters required? Accuracy being equal for that object type.

Thank You for your content.

Tom

Jason Brownlee October 24, 2020 at 7:10 am # REPLY ↰

Yes, we call it the capacity of the model. It's related (a complex relationship to be sure) to the difficulty of the modeling/prediction task.

This might be relevant:
https://machinelearningmastery.com/how-to-control-neural-network-model-capacity-with-nodes-and-layers/

Turyal November 21, 2020 at 12:19 am # REPLY ↰

Hi Jason,

what will be the appropriate number of filters using 3 x 3 filter in conv layer for 224 x 224 x 3 input image?

Jason Brownlee November 21, 2020 at 6:42 am # REPLY ↰

There is no best number, try different values and discover what works well/best for your specific model and dataset.

garima December 4, 2020 at 10:39 pm # REPLY ↰

hi Jason,
can you please explain to me how the value of the filter gets decided? is CNN randomly taking it and then after weight updation its value get updated , and can we take design its value as per our requirement? please reply

Jason Brownlee December 5, 2020 at 8:06 am # REPLY ↰

The number of layers and number of filters can be chosen a number of ways, see this:
https://machinelearningmastery.com/faq/single-faq/how-many-layers-and-nodes-do-i-need-in-my-neural-network

Aditya Raj Singh December 18, 2020 at 3:32 pm # REPLY ↰

That made a job so much easier for me to implement;

Thanks in advance Jason;

Please continue doing the good work, your articles are so interesting and knowledgeable 😉

Jason Brownlee December 19, 2020 at 6:14 am # REPLY ↰

You're welcome. I'm happy to hear that.

Dr khan January 7, 2021 at 2:34 am #

REPLY

Grest work , much appreciated

Jason Brownlee January 7, 2021 at 6:20 am #

REPLY

Thanks!

Esther January 7, 2021 at 5:24 pm #

REPLY

Sir
Why is the filter in convolution layer called a learnable filter.

The kernel initial values are random and it extracts the features. Where is the learning taking place. where is the updating of filter value taking place.

I am really confused

Jason Brownlee January 8, 2021 at 5:40 am #

REPLY

Because the model parameters called weights are adapted based on the training data.

Noura February 6, 2021 at 3:44 am #

REPLY

Hi sir, Thanks for the great tutorial.
I have two questions please.

-How the feature maps are connected in two different convolutional layers. Say we have first conv layer with 10 filters, and second conv layer with 64 filtres. The second layer is used directly after the first layer. So we have 10 feature maps as the output of the first conv layer that are passed to the next layer which is supposed to produce 64 feature maps. I wonder what is the relation between the 10 feature maps (the input of the second layer) and the 64 feature maps(the output of the same layer)?

-What is the effect of using Dropout between conv layers? does it drop feature maps? I use it in my project and it seems to perform well, but I don't understand why.

Thanks a lot for your help.

Jason Brownlee February 6, 2021 at 5:57 am #

REPLY

You're welcome.

The output of one convolutional layer will be a number of feature maps. These are matrices of numbers or "images" that can be fed into the next set of convolutional layers directly – just like we fed an image into the first convolutional layer.

Typically we have pooling layers sitting between convolutional layers to make the matrices (images) smaller on the way through.

Dropout is a type of regularization and helps us slow down learning and make the model more general (better prediction on unseen data):
https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

Noura February 6, 2021 at 8:30 am # REPLY ↰

Great. Thanks for your answer.
I still don't understand how the dropout works in between convolutional layers. I know that it drops neurons in dense layers, but since we have feature maps that are passed between conv layers, I don't really get how it operates on those feature maps. Could you please clarify it for me? or recommend some useful resources?

Thanks again for your help

martin February 6, 2021 at 4:22 am # REPLY ↰

Thanks for the tutorial!

Jason Brownlee February 6, 2021 at 5:57 am # REPLY ↰

You're welcome!

Jam February 19, 2021 at 12:31 am # REPLY ↰

Thank you so much – this is excellent! I love your stuff. Keep it up!

Jason Brownlee February 19, 2021 at 6:00 am # REPLY ↰

Thanks!

Ayush Khare March 23, 2021 at 9:33 pm # REPLY ↰

You have got an excellent understanding of what the readers would want and where should some extra emphasis be applied, so as the readers could understand better. I mean its in every way apt to understand the mentioned concept and get intuitions. Thanks a lot for this great work.

Jason Brownlee March 24, 2021 at 5:51 am # REPLY ↰

Thanks!

Hesham Ali May 8, 2021 at 8:16 am # REPLY

thank you Jason, you always have my back.
I have one question after max poling the matrix flattened to enter neural nets so how backpropagation happens in CNN like how kernels updated.

Jason Brownlee May 9, 2021 at 5:51 am # REPLY

Good question and a great suggestion for a future blog post tutorial! Thanks.

Vedika Hatekar May 16, 2021 at 12:22 am # REPLY

Hi Jason. You gave an excellent explanation of CNN. Thanks a lot! I have a small question. How should we decide the shape of the filter and also number of filters needed?

Jason Brownlee May 16, 2021 at 5:34 am # REPLY

You're welcome.

You can use trial and error to determine the shape and number of filters. Or copy values used in another model as a starting point.

Alexander Perry August 5, 2021 at 2:57 am # REPLY

As usual, excellent article. Thank you for taking the time to share your knowledge!

Jason Brownlee August 5, 2021 at 5:25 am # REPLY

Thanks!

mhr August 23, 2021 at 4:53 pm # REPLY

An amazing article . thanks.

Adrian Tam August 24, 2021 at 8:22 am # REPLY

Thanks.

Ben October 8, 2021 at 11:54 pm # REPLY

Hey Jason,

First of all, thank you so much for all the great content! I really like your explanations and have learned so much just from reading your posts!

I had a question regarding filters. You mention that a convolutional layer will use many filters (e.g. 32, 64, 128) but I don't really understand how those filters would differ. Could you explain how the filters would learn different features, despite being applied to the same image?

Thanks!

Adrian Tam October 13, 2021 at 5:33 am #

REPLY ↰

Assume you have an image of fairly large size. One feature you may want to find is the bright spots on the image. You can use a 32×32 filter and average pooling to find the average brightness in any 32×32 region, or a 128×128 filter in 128×128 region. Hence the filter size will give you a different field of view.

Ben October 14, 2021 at 12:13 am #

REPLY ↰

Hey Jason,

Thanks for your reply! Your answer sounds like we are applying a single feature to the image, whether it is 3×3 or 32×32, etc. However, I was under the impression that a layer like:

model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(200, 200, 3)))

would create many filters (32 in this case). Is that true? And if it is, my question is more around how these features are different from each other even though they are applied to the same image.

Thanks again! Really appreciate all the content!

Ben October 14, 2021 at 12:14 am #

REPLY ↰

*how these filters are different from each other

Adrian Tam October 14, 2021 at 3:25 am #

REPLY ↰

Your understand is correct. For how different filters are different from each other, I don't know. But when you train the network, probably it will magically converge to something different (e.g., one pick up the horizontal edge, and the other pick up vertical edge). May be you would like to read this post: https://machinelearningmastery.com/how-to-visualize-filters-and-feature-maps-in-convolutional-neural-networks/

Ben October 15, 2021 at 12:29 pm #

REPLY ↰

That was very interesting. Thank you!

diospyros November 2, 2022 at 4:11 am #

REPLY ↰

Haven't seen this stated explicitly anywhere and was hoping you might be able to provide clarity. Say I have a bunch of 64,64,3 images but instead of having them stacked like :,64,64,3, I have them organized into higher dimensions for modeling purposes, for instance, (:,5,5,5,64,64,3). Then I run a 2D convolution (tensorflow) with stride=1 and get (:,5,5,5,64,64,1). Can I assume that the convolution iterated over all 5x5x5=125 instances of 64x64x3 images per batch sample; thus intuitively maintaining the integrity of my structure? My hope is that it just defaults to the three highest dimensions and is smart enough to iterate over all others.

Python is a bit opaque to me, so I'm a little nervous about making assumptions.

James Carmichael November 2, 2022 at 6:48 am #

REPLY

Hi diospyros…You may find the following resource of interest to build Python skills.

https://machinelearningmastery.com/start-here/#pythonskills

diospyros November 3, 2022 at 12:40 am #

REPLY

Thanks for the link. I'm pretty functional in Python, though. I'm just used to being able to see the underlying data structures at the address and bit level in C++/C when I want to understand how a library is working. 😉

More interested in the assumptions that TensorFlow is making under the hood (or at least not clearly documented). Guess I can just Reshape( ) the lower dims but was wondering if was unnecessary.

Leave a Reply

Name (required)

Email (will not be published) (required)

SUBMIT COMMENT

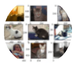
Welcome!
I'm *Jason Brownlee* PhD
and I **help developers** get results with **machine learning**.
Read more

Never miss a tutorial:

Picked for you:

How to Train an Object Detection Model with Keras

How to Develop a Face Recognition System Using FaceNet in Keras


How to Classify Photos of Dogs and Cats (with 97% accuracy)

How to Perform Object Detection With YOLOv3 in Keras

How to Get Started With Deep Learning for Computer Vision (7-Day Mini-Course)

LinkedIn | Twitter | Facebook | Newsletter | RSS

Privacy | Disclaimer | Terms | Contact | Sitemap | Search

EXHIBIT K

Published in Towards Data Science

George Seif
May 14, 2019 · 5 min read · Member-only · Listen

Visualising Filters and Feature Maps for Deep Learning

> *Want to be inspired? Come join my* ***Super Quotes newsletter***. 😎

Deep Neural Networks are one of the most powerful class of machine learning models. With enough data, their accuracy in tasks such as Computer Vision and Natural Language Processing (NLP) is unmatched.

The only drawback that many scientists will comment on is the fact that these networks are completely black-box. We still have very little knowledge as to how deep networks learn their target patterns so well, especially how all the neurons work together to achieve the final result.

Having some kind of inside knowledge of how an ML model works presents a few key advantages:

- They're easier to tune, since when we see an error made by the network we can point out the cause directly
- The functionality and expected behaviour of the networks can be explained, especially to non-technical stakeholders
- We can further extend and improve the overall design of our models since we'd have knowledge of the current design, including the strengths and weaknesses of it

The great thing is that researchers have begun to explore techniques for understanding what goes on inside deep networks. In particular, we now have the ability to visualise the filters that Convolutional Neural Networks (CNNs) learn from their training for Computer Vision tasks, and the feature maps that the CNN produces when applied to an image.

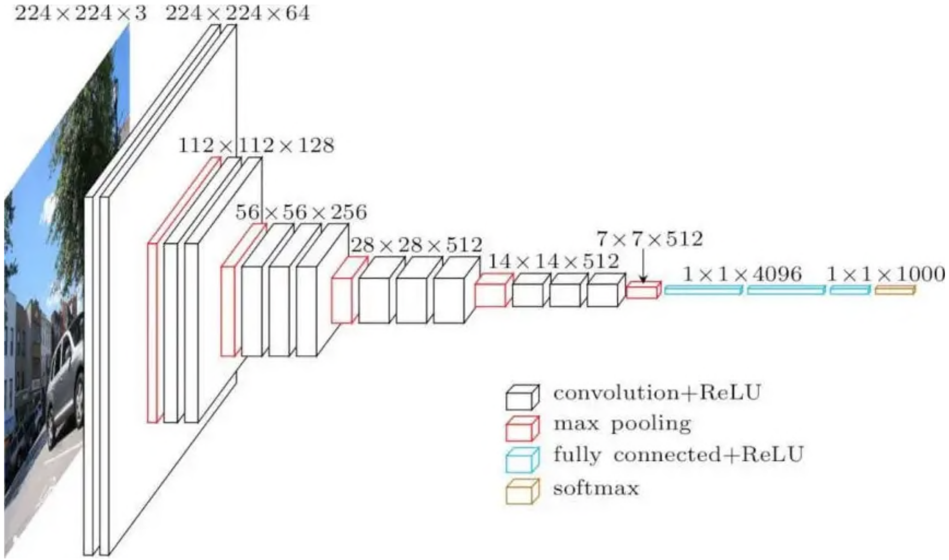
This tutorial will teach you how to do just that: visualise the filters and feature maps of a CNN trained for image classification. We'll even do it in the super easy to use Keras!

Picking Our Model

First we'll need to pick a model to visualise.

The easiest way to do this in Keras is to select a pre-trained image classification model. Keras lots of these top-performing models in its Applications page. All of them were trained on the ImageNet dataset with over a million images.

For this tutorial, I'm going to pick out the VGG16 Model. It's simple enough for us to go through our tutorial and still holds its own as a fairly accurate model. The structure is shown below.



The process of loading up the model in Keras is shown below. When you import the model from `keras.applications`, Keras will automatically download the weights for you in the appropriate directory. We then load the model and print out a summary of its structure.

```python
from keras.applications.vgg16 import VGG16

# Load up our model
model = VGG16()

# Print out a summary of our model
model.summary()
```

load_vgg16.py hosted with ❤ by GitHub view raw

Visualising Filters

The first visualisation we'll create is that of the CNN filters.

When Deep Learning folks talk about "filters" what they're referring to is the learned weights of the convolutions. For example, a single 3x3 convolution is called a "filter" and that filter has a total of 10 weights (9 + 1 bias).

By visualising the learned weights we can get some idea as to how well our network has learned. For example, if we see a lot of zeros then we'll know we have many dead filters that aren't going much for our network — a great opportunity to do some pruning for model compression.

Take a look at the code below for visualising the filters and then scroll down for an explanation of how it works.

```python
from keras.applications.vgg16 import VGG16
import matplotlib.pyplot as plt
import numpy as np

model = VGG16()

layer_dict = dict([(layer.name, layer) for layer in model.layers])

layer_name = 'block5_conv1'
filter_index = 0 # Which filter in this block would you like to visualise?

# Grab the filters and biases for that layer
filters, biases = layer_dict[layer_name].get_weights()

# Normalize filter values to a range of 0 to 1 so we can visualize them
f_min, f_max = np.amin(filters), np.amax(filters)
filters = (filters - f_min) / (f_max - f_min)

# Plot first few filters
n_filters, index = 6, 1
for i in range(n_filters):
    f = filters[:, :, :, i]

    # Plot each channel separately
    for j in range(3):

        ax = plt.subplot(n_filters, 3, index)
        ax.set_xticks([])
        ax.set_yticks([])

        plt.imshow(f[:, :, j], cmap='viridis')
        index += 1

plt.show()
```

visualise_filters.py hosted with ❤ by GitHub view raw

After importing all of our necessary libraries and loading up our model, we build a dictionary of the layers in our model at line 7. This will allow us to access the weights at each layer by the layer name. You can print out the keys in the dictionary if you want to check out all the names.

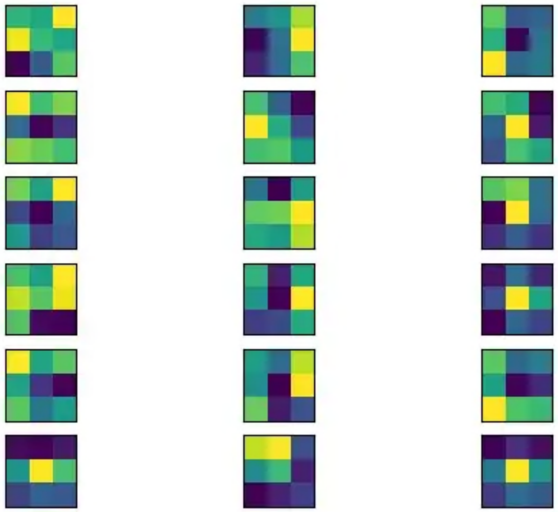
Next we'll go ahead and set the name of the layer whose filters we want to visualise in line 9. I picked the "block5_conv1" but you can select any of the convolution blocks you like.

In line 13 we call the `get_weights()` function to grab the filter and bias weights from our selected layer.

In lines 16 and 17 we normalise the filter values such that they're between 0 and 1. This will help create a clear visualisation when we show the weights as colours on the screen.

Finally, our loop starting at line 21 displays some of the filters. For the "viridis" colour map we selected, yellow represents a value of 1 and dark blue a value of 0.

Check out the output below for VGG16's block5_conv1 layer!



Visualising Feature Maps

The feature maps of a CNN capture the result of applying the filters to an input image. I.e at each layer, the feature map is the output of that layer.

The reason for visualising a feature map for a specific input image is to try to gain some understanding of what features our CNN detects. Perhaps it detects some parts of our desired object and not others or the activations die out at a certain layer.

It is also interesting to directly observe a common idea in Deep Learning, that the early layers of the network detect low-level features (colours, edges, etc) and the later layers of the network detect high-level features (shapes and objects).

Take a look at the code below for visualising the feature maps and then scroll down for an explanation of how it works.

```python
from keras.applications.vgg16 import VGG16, preprocess_input
from keras.preprocessing.image import load_img,img_to_array
from keras.models import Model
import matplotlib.pyplot as plt
import numpy as np

model = VGG16()

layer_dict = dict([(layer.name, layer) for layer in model.layers])

layer_name = 'block1_conv2'

model = Model(inputs=model.inputs, outputs=layer_dict[layer_name].output)

# Perpare the image
image = load_img('tiger.jpg', target_size=(224, 224))
image = img_to_array(image)
image = np.expand_dims(image, axis=0)
image = preprocess_input(image)

# Apply the model to the image
feature_maps = model.predict(image)

square = 8
index = 1
for _ in range(square):
        for _ in range(square):

                ax = plt.subplot(square, square, index)
                ax.set_xticks([])
                ax.set_yticks([])

                plt.imshow(feature_maps[0, :, :, index-1], cmap='viridis')
                index += 1

plt.show()
```

visualise_feature_maps.py hosted with ❤ by GitHub view raw

The first couple of steps remain the same: load up the model and select the name of the layer you want to visualise. The one minor change is that we modify our model such that the final output is the output of the VGG16's last feature map, since we don't really need the last couple of fully connected layers.

Next we'll load up and prepare our image in lines 16 to 19. You can choose any image you like; I chose an image of a tiger because tigers are awesome.

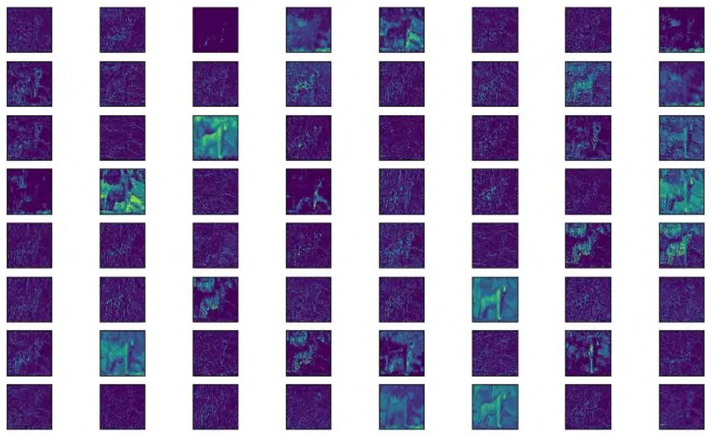
That code in lines 16 to 19 will read in the image and apply the necessary pre-processing that the VGG16 model requires. Don't worry if your image is a huge size like 4K resolution because VGG always expects an image of size 224x224, so the input will always be re-sized before processing.

At line 22 we apply our VGG16 model to the input image; the output will be our desired feature map. We'll loop through a few of the channels of the map and visualise them in a matplotlib figure.

Check out the resulting figure below. Notice how there's many activations on the edges and textures within the image, especially the outline of our tiger!

Get started

Sign In



. . .

Like to learn?

Follow me on twitter where I post all about the latest and greatest AI, Technology, and Science! Connect with me on LinkedIn too!

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. Take a look.

By signing up, you will create a Medium account if you don't already have one. Review our Privacy Policy for more information about our privacy practices.

Get this newsletter

Search

George Seif

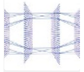
21K Followers

Follow

More from Medium

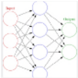
Dharmaraj

Convolutional Neural Networks (CNN)—...



Roopa CM

Activation Functions in Artificial Neural...



Ruks... in Towar...

Convolutional Neural Network (CNN)...

Ke... in Dev Gen...

Semi-supervised learning with autoencoders

Help Status Writers Blog Careers
Privacy Terms About Text to speech

EXHIBIT L

Machine Learning Mastery
Making Developers Awesome at Machine Learning

Search...

A Gentle Introduction to Object Recognition With Deep Learning

by **Jason Brownlee** on May 22, 2019 in **Deep Learning for Computer Vision**

Tweet Tweet Share Share

Last Updated on January 27, 2021

It can be challenging for beginners to distinguish between different related computer vision tasks.

For example, image classification is straight forward, but the differences between object localization and object detection can be confusing, especially when all three tasks may be just as equally referred to as object recognition.

Image classification involves assigning a class label to an image, whereas object localization involves drawing a bounding box around one or more objects in an image. Object detection is more challenging and combines these two tasks and draws a bounding box around each object of interest in the image and assigns them a class label. Together, all of these problems are referred to as object recognition.

In this post, you will discover a gentle introduction to the problem of object recognition and state-of-the-art deep learning models designed to address it.

After reading this post, you will know:

- Object recognition is refers to a collection of related tasks for identifying objects in digital photographs.
- Region-Based Convolutional Neural Networks, or R-CNNs, are a family of techniques for addressing object localization and recognition tasks, designed for model performance.
- You Only Look Once, or YOLO, is a second family of techniques for object recognition designed for speed and real-time use.

Kick-start your project with my new book Deep Learning for Computer Vision, including *step-by-step tutorials* and the *Python source code* files for all examples.

Let's get started.

Never miss a tutorial:

Picked for you:

How to Train an Object Detection Model with Keras

How to Develop a Face Recognition System Using FaceNet in Keras

How to Classify Photos of Dogs and Cats (with 97% accuracy)

How to Perform Object Detection With YOLOv3 in Keras

How to Get Started With Deep Learning for Computer Vision (7-Day Mini-Course)

A Gentle Introduction to Object Recognition With Deep Learning
Photo by Bart Everson, some rights reserved.

Overview

This tutorial is divided into three parts; they are:

1. What is Object Recognition?
2. R-CNN Model Family
3. YOLO Model Family

What is Object Recognition?

Object recognition is a general term to describe a collection of related computer vision tasks that involve identifying objects in digital photographs.

Image classification involves predicting the class of one object in an image. *Object localization* refers to identifying the location of one or more objects in an image and drawing abounding box around their extent. *Object detection* combines these two tasks and localizes and classifies one or more objects in an image.

When a user or practitioner refers to "*object recognition*", they often mean "*object detection*".

> *… we will be using the term object recognition broadly to encompass both image classification (a task requiring an algorithm to determine what object classes are present in the image) as well as object detection (a task requiring an algorithm to localize all objects present in the image*

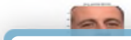
— ImageNet Large Scale Visual Recognition Challenge, 2015.

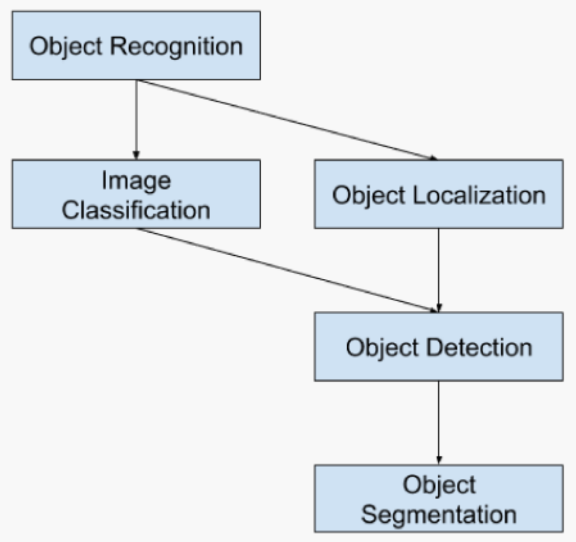
As such, we can distinguish between these three computer vision tasks:

- **Image Classification**: Predict the type or class of an object in an image.
  - *Input*: An image with a single object, such as a photograph.
  - *Output*: A class label (e.g. one or more integers that are mapped to class labels).
- **Object Localization**: Locate the presence of objects in an image and indicate their location with a bounding box.
  - *Input*: An image with one or more objects, such as a photograph.
  - *Output*: One or more bounding boxes (e.g. defined by a point, width, and height).
- **Object Detection**: Locate the presence of objects with a bounding box and types or classes of the located objects in an image.
  - *Input*: An image with one or more objects, such as a photograph.
  - *Output*: One or more bounding boxes (e.g. defined by a point, width, and height), and a class label for each bounding box.

One further extension to this breakdown of computer vision tasks is *object segmentation*, also called "object instance segmentation" or "semantic segmentation," where instances of recognized objects are indicated by highlighting the specific pixels of the object instead of a coarse bounding box.

From this breakdown, we can see that object recognition refers to a suite of challenging computer vision tasks.





Overview of Object Recognition Computer Vision Tasks

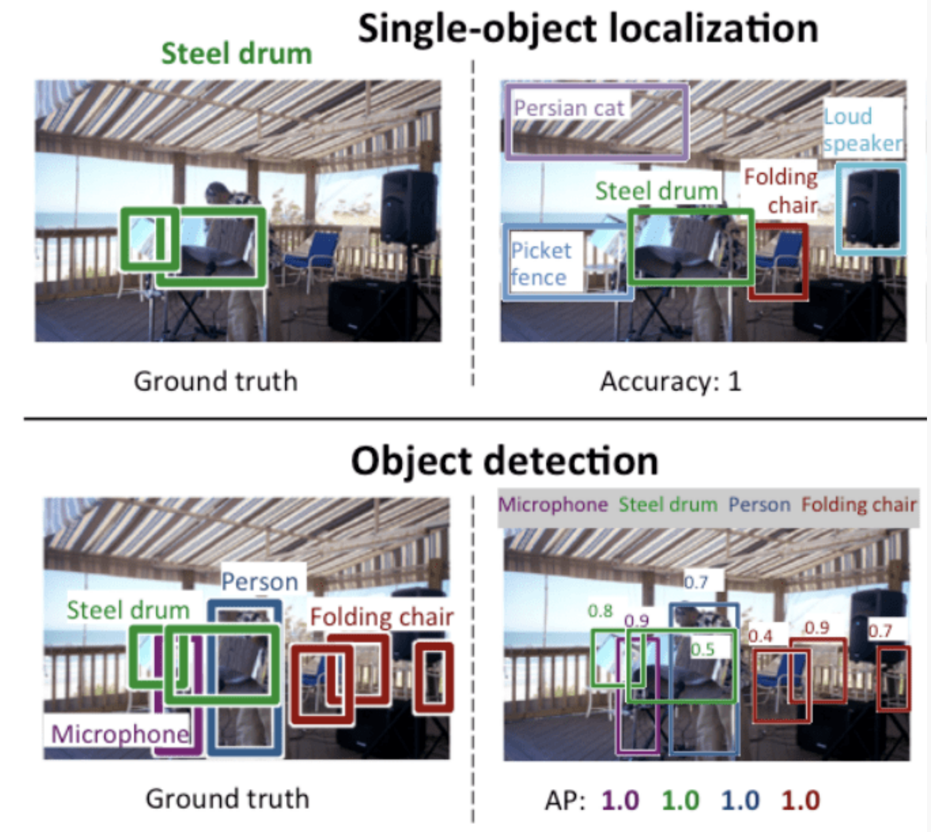
Most of the recent innovations in image recognition problems have come as part of participation in the ILSVRC tasks.

This is an annual academic competition with a separate challenge for each of these three problem types, with the intent of fostering independent and separate improvements at each level that can be leveraged more broadly. For example, see the list of the three corresponding task types below taken from the 2015 ILSVRC review paper:

- **Image classification**: Algorithms produce a list of object categories present in the image.
- **Single-object localization**: Algorithms produce a list of object categories present in the image, along with an axis-aligned bounding box indicating the position and scale of one instance of each object category.
- **Object detection**: Algorithms produce a list of object categories present in the image along with an axis-aligned bounding box indicating the position and scale of every instance of each object category.

We can see that "*Single-object localization*" is a simpler version of the more broadly defined "*Object Localization*," constraining the localization tasks to objects of one type within an image, which we may assume is an easier task.

Below is an example comparing single object localization and object detection, taken from the ILSVRC paper. Note the difference in ground truth expectations in each case.



Comparison Between Single Object Localization and Object Detection.Taken From: ImageNet Large Scale Visual Recognition Challenge.

The performance of a model for image classification is evaluated using the mean classification error across the predicted class labels. The performance of a model for single-object localization is evaluated using the distance between the expected and predicted bounding box for the expected class. Whereas the performance of a model for object recognition is evaluated using the precision and recall across each of the best matching bounding boxes for the known objects in the image.

Now that we are familiar with the problem of object localization and detection, let's take a look at some recent top-performing deep learning models.

R-CNN Model Family

The R-CNN family of methods refers to the R-CNN, which may stand for "*Regions with CNN Features*" or "*Region-Based Convolutional Neural Network*," developed by Ross Girshick, et al.

This includes the techniques R-CNN, Fast R-CNN, and Faster-RCNN designed and demonstrated for object localization and object recognition.

Let's take a closer look at the highlights of each of these techniques in turn.

R-CNN

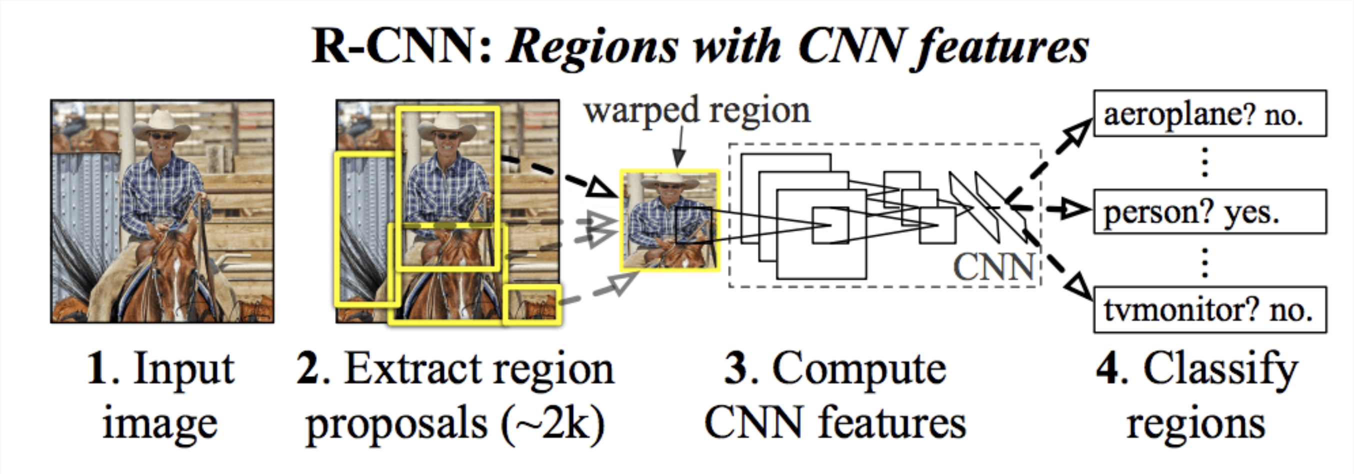
The R-CNN was described in the 2014 paper by Ross Girshick, et al. from UC Berkeley titled "Rich feature hierarchies for accurate object detection and semantic segmentation."

It may have been one of the first large and successful application of convolutional neural networks to the problem of object localization, detection, and segmentation. The approach was demonstrated on benchmark datasets, achieving then state-of-the-art results on the VOC-2012 dataset and the 200-class ILSVRC-2013 object detection dataset.

Their proposed R-CNN model is comprised of three modules; they are:

- **Module 1: Region Proposal**. Generate and extract category independent region proposals, e.g. candidate bounding boxes.
- **Module 2: Feature Extractor**. Extract feature from each candidate region, e.g. using a deep convolutional neural network.
- **Module 3: Classifier**. Classify features as one of the known class, e.g. linear SVM classifier model.

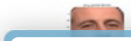
The architecture of the model is summarized in the image below, taken from the paper.



Summary of the R-CNN Model ArchitectureTaken from Rich feature hierarchies for accurate object detection and semantic segmentation.

A computer vision technique is used to propose candidate regions or bounding boxes of potential objects in the image called "*selective search*," although the flexibility of the design allows other region proposal algorithms to be used.

The feature extractor used by the model was the AlexNet deep CNN that won the ILSVRC-2012 image classification competition. The output of the CNN was a 4,096 element vector that describes the contents of the image that is fed to a linear SVM for classification, specifically one SVM is trained for each known class.



It is a relatively simple and straightforward application of CNNs to the problem of object localization and recognition. A downside of the approach is that it is slow, requiring a CNN-based feature extraction pass on each of the candidate regions generated by the region proposal algorithm. This is a problem as the paper describes the model operating upon approximately 2,000 proposed regions per image at test-time.

Python (Caffe) and MatLab source code for R-CNN as described in the paper was made available in the R-CNN GitHub repository.

Fast R-CNN

Given the great success of R-CNN, Ross Girshick, then at Microsoft Research, proposed an extension to address the speed issues of R-CNN in a 2015 paper titled "Fast R-CNN."

The paper opens with a review of the limitations of R-CNN, which can be summarized as follows:

- **Training is a multi-stage pipeline**. Involves the preparation and operation of three separate models.
- **Training is expensive in space and time**. Training a deep CNN on so many region proposals per image is very slow.
- **Object detection is slow**. Make predictions using a deep CNN on so many region proposals is very slow.

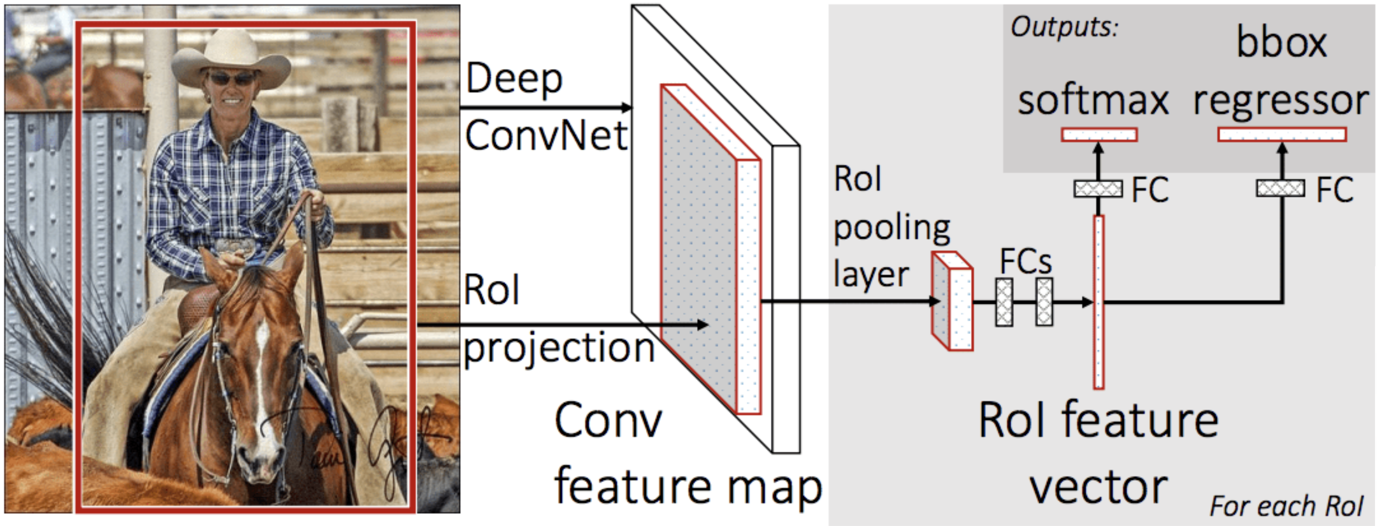
A prior work was proposed to speed up the technique called spatial pyramid pooling networks, or SPPnets, in the 2014 paper "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition." This did speed up the extraction of features, but essentially used a type of forward pass caching algorithm.

Fast R-CNN is proposed as a single model instead of a pipeline to learn and output regions and classifications directly.

The architecture of the model takes the photograph a set of region proposals as input that are passed through a deep convolutional neural network. A pre-trained CNN, such as a VGG-16, is used for feature extraction. The end of the deep CNN is a custom layer called a Region of Interest Pooling Layer, or RoI Pooling, that extracts features specific for a given input candidate region.

The output of the CNN is then interpreted by a fully connected layer then the model bifurcates into two outputs, one for the class prediction via a softmax layer, and another with a linear output for the bounding box. This process is then repeated multiple times for each region of interest in a given image.

The architecture of the model is summarized in the image below, taken from the paper.



Summary of the Fast R-CNN Model Architecture.
Taken from: Fast R-CNN.

The model is significantly faster to train and to make predictions, yet still requires a set of candidate regions to be proposed along with each input image.

Python and C++ (Caffe) source code for Fast R-CNN as described in the paper was made available in a GitHub repository.

Faster R-CNN

The model architecture was further improved for both speed of training and detection by Shaoqing Ren, et al. at Microsoft Research in the 2016 paper titled "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks."

The architecture was the basis for the first-place results achieved on both the ILSVRC-2015 and MS COCO-2015 object recognition and detection competition tasks.

The architecture was designed to both propose and refine region proposals as part of the training process, referred to as a Region Proposal Network, or RPN. These regions are then used in concert with a Fast R-CNN model in a single model design. These improvements both reduce the number of region proposals and accelerate the test-time operation of the model to near real-time with then state-of-the-art performance.

> *… our detection system has a frame rate of 5fps (including all steps) on a GPU, while achieving state-of-the-art object detection accuracy on PASCAL VOC 2007, 2012, and MS COCO datasets with only 300 proposals per image. In ILSVRC and COCO 2015 competitions, Faster R-CNN and RPN are the foundations of the 1st-place winning entries in several tracks*

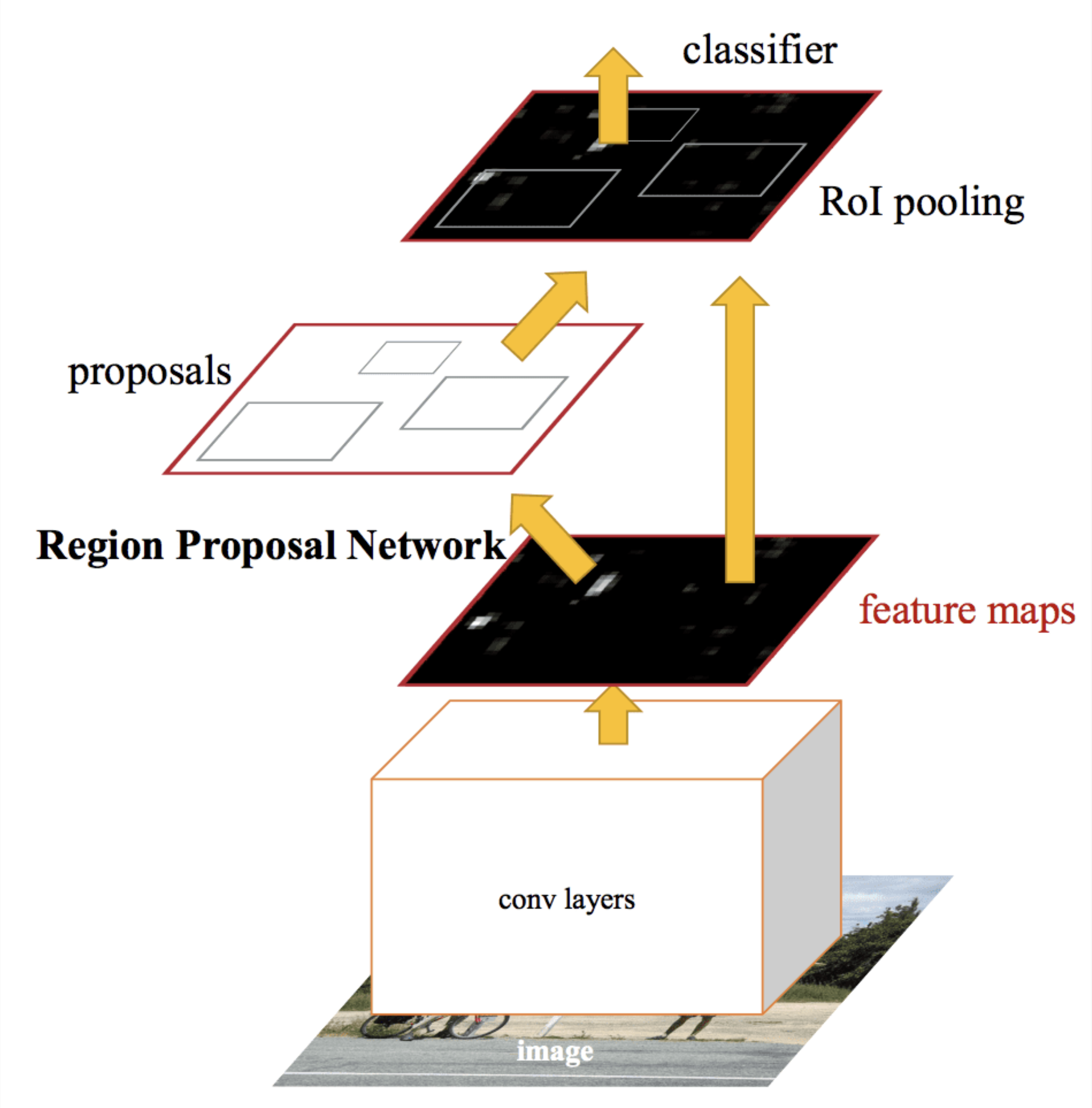
— Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, 2016.

Although it is a single unified model, the architecture is comprised of two modules:

- **Module 1: Region Proposal Network**. Convolutional neural network for proposing regions and the type of object to consider in the region.
- **Module 2: Fast R-CNN**. Convolutional neural network for extracting features from the proposed regions and outputting the bounding box and class labels.

Both modules operate on the same output of a deep CNN. The region proposal network acts as an attention mechanism for the Fast R-CNN network, informing the second network of where to look or pay attention.

The architecture of the model is summarized in the image below, taken from the paper.



Summary of the Faster R-CNN Model Architecture.Taken from: Faster R-CNN: Towards Real-Time Object Detection With Region Proposal Networks.

The RPN works by taking the output of a pre-trained deep CNN, such as VGG-16, and passing a small network over the feature map and outputting multiple region proposals and a class prediction for each. Region proposals are bounding boxes, based on so-called anchor boxes or pre-defined shapes designed to accelerate and improve the proposal of regions. The class prediction is binary, indicating the presence of an object, or not, so-called "*objectness*" of the proposed region.

A procedure of alternating training is used where both sub-networks are trained at the same time, although interleaved. This allows the parameters in the feature detector deep CNN to be tailored or fine-tuned for both tasks at the same time.

At the time of writing, this Faster R-CNN architecture is the pinnacle of the family of models and continues to achieve near state-of-the-art results on object recognition tasks. A further extension adds support for image segmentation, described in the paper 2017 paper "Mask R-CNN."

Python and C++ (Caffe) source code for Fast R-CNN as described in the paper was made available in a GitHub repository.

YOLO Model Family

Another popular family of object recognition models is referred to collectively as YOLO or "*You Only Look Once*," developed by Joseph Redmon, et al.

The R-CNN models may be generally more accurate, yet the YOLO family of models are fast, much faster than R-CNN, achieving object detection in real-time.

YOLO

The YOLO model was first described by Joseph Redmon, et al. in the 2015 paper titled "You Only Look Once: Unified, Real-Time Object Detection." Note that Ross Girshick, developer of R-CNN, was also an author and contributor to this work, then at Facebook AI Research.

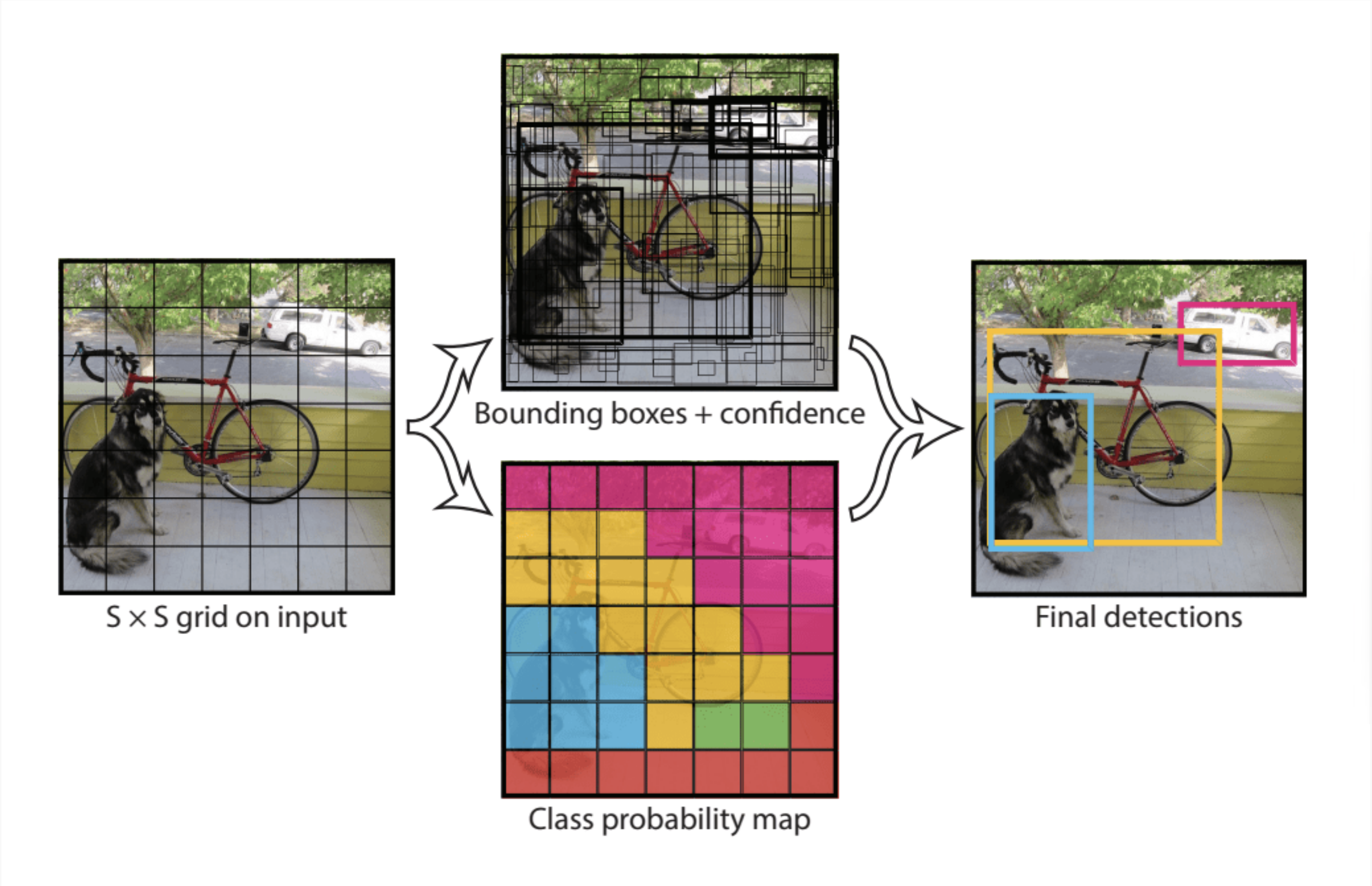
The approach involves a single neural network trained end to end that takes a photograph as input and predicts bounding boxes and class labels for each bounding box directly. The technique offers lower predictive accuracy (e.g. more localization errors), although operates at 45 frames per second and up to 155 frames per second for a speed-optimized version of the model.

> *Our unified architecture is extremely fast. Our base YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second …*

— You Only Look Once: Unified, Real-Time Object Detection, 2015.

The model works by first splitting the input image into a grid of cells, where each cell is responsible for predicting a bounding box if the center of a bounding box falls within the cell. Each grid cell predicts a bounding box involving the x, y coordinate and the width and height and the confidence. A class prediction is also based on each cell.

For example, an image may be divided into a 7×7 grid and each cell in the grid may predict 2 bounding boxes, resulting in 94 proposed bounding box predictions. The class probabilities map and the bounding boxes with confidences are then combined into a final set of bounding boxes and class labels. The image taken from the paper below summarizes the two outputs of the model.



Summary of Predictions made by YOLO Model.Taken from: You Only Look Once: Unified, Real-Time Object Detection

YOLOv2 (YOLO9000) and YOLOv3

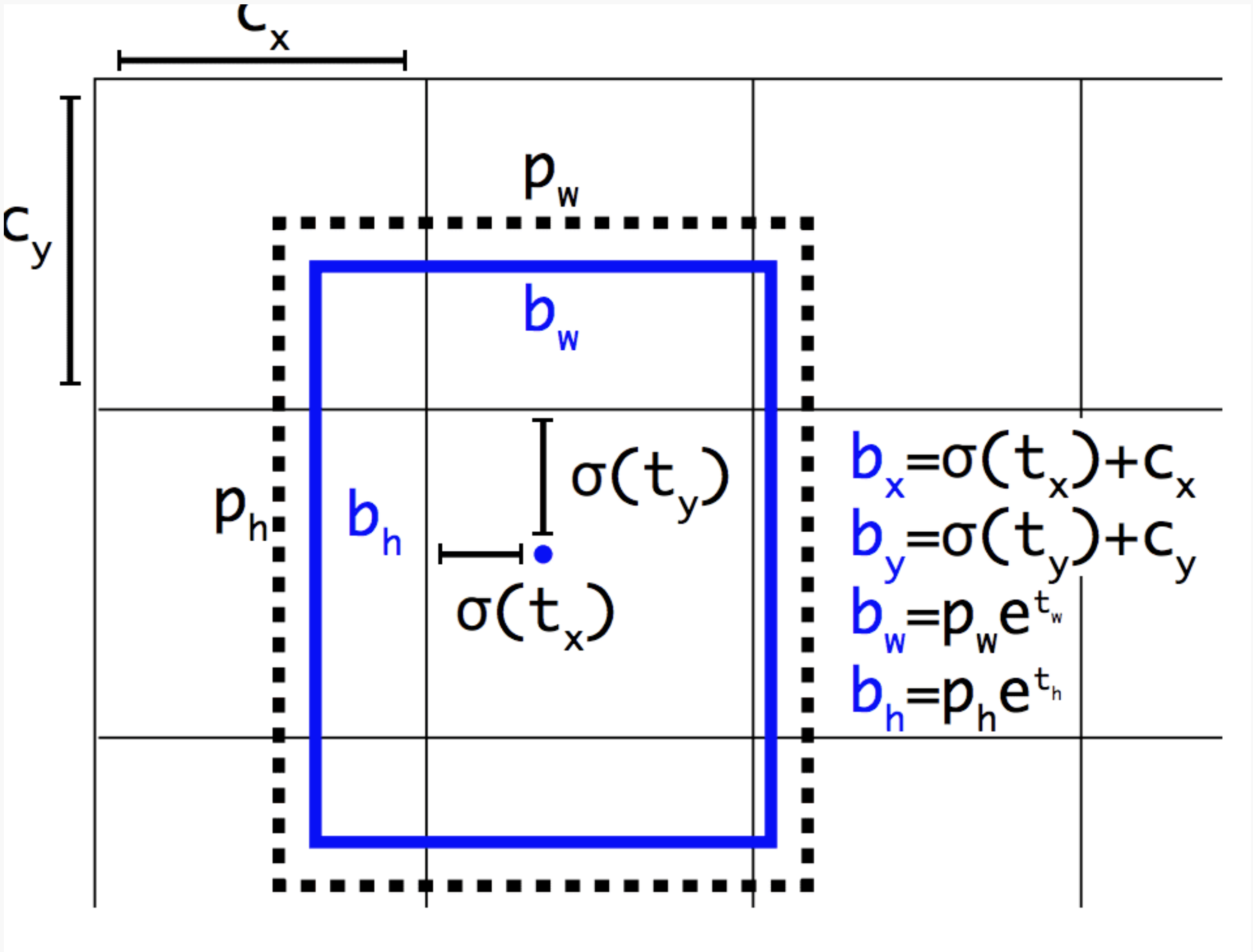
The model was updated by Joseph Redmon and Ali Farhadi in an effort to further improve model performance in their 2016 paper titled "YOLO9000: Better, Faster, Stronger."

Although this variation of the model is referred to as YOLO v2, an instance of the model is described that was trained on two object recognition datasets in parallel, capable of predicting 9,000 object classes, hence given the name "*YOLO9000*."

A number of training and architectural changes were made to the model, such as the use of batch normalization and high-resolution input images.

Like Faster R-CNN, YOLOv2 model makes use of anchor boxes, pre-defined bounding boxes with useful shapes and sizes that are tailored during training. The choice of bounding boxes for the image is pre-processed using a k-means analysis on the training dataset.

Importantly, the predicted representation of the bounding boxes is changed to allow small changes to have a less dramatic effect on the predictions, resulting in a more stable model. Rather than predicting position and size directly, offsets are predicted for moving and reshaping the pre-defined anchor boxes relative to a grid cell and dampened by a logistic function.



Example of the Representation Chosen when Predicting Bounding Box Position and ShapeTaken from: YOLO9000: Better, Faster, Stronger

Never miss a tutorial:

Picked for you:

How to Train an Object Detection Model with Keras

How to Develop a Face Recognition

Further improvements to the model were proposed by Joseph Redmon and Ali Farhadi in their 2018 paper titled "YOLOv3: An Incremental Improvement." The improvements were reasonably minor, including a deeper feature detector network and minor representational changes.

Further Reading

This section provides more resources on the topic if you are looking to go deeper.

Papers

- ImageNet Large Scale Visual Recognition Challenge, 2015.

R-CNN Family Papers

- Rich feature hierarchies for accurate object detection and semantic segmentation, 2013.
- Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition, 2014.
- Fast R-CNN, 2015.
- Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, 2016.
- Mask R-CNN, 2017.

YOLO Family Papers

- You Only Look Once: Unified, Real-Time Object Detection, 2015.
- YOLO9000: Better, Faster, Stronger, 2016.
- YOLOv3: An Incremental Improvement, 2018.

Code Projects

- R-CNN: Regions with Convolutional Neural Network Features, GitHub.
- Fast R-CNN, GitHub.
- Faster R-CNN Python Code, GitHub.
- YOLO, GitHub.

Resources

- Ross Girshick, Homepage.
- Joseph Redmon, Homepage.
- YOLO: Real-Time Object Detection, Homepage.

Articles

- A Brief History of CNNs in Image Segmentation: From R-CNN to Mask R-CNN, 2017.
- Object Detection for Dummies Part 3: R-CNN Family, 2017.
- Object Detection Part 4: Fast Detection Models, 2018.

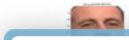
Summary

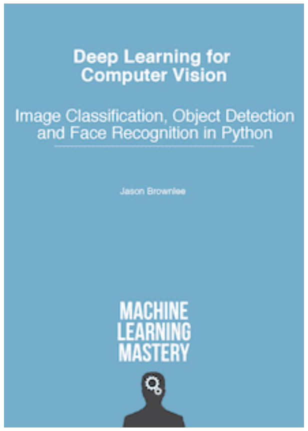
In this post, you discovered a gentle introduction to the problem of object recognition and state-of-the-art deep learning models designed to address it.

Specifically, you learned:

- Object recognition is refers to a collection of related tasks for identifying objects in digital photographs.
- Region-Based Convolutional Neural Networks, or R-CNNs, are a family of techniques for addressing object localization and recognition tasks, designed for model performance.
- You Only Look Once, or YOLO, is a second family of techniques for object recognition designed for speed and real-time use.

Do you have any questions?
Ask your questions in the comments below and I will do my best to answer.




Deep Learning for Computer Vision
Image Classification, Object Detection and Face Recognition in Python
Jason Brownlee
MACHINE LEARNING MASTERY

Discover how in my new Ebook:
Deep Learning for Computer Vision

It provides **self-study tutorials** on topics like:
classification, object detection (yolo and rcnn), face recognition (vggface and facenet), data preparation and much more...

Finally Bring Deep Learning to your Vision Projects

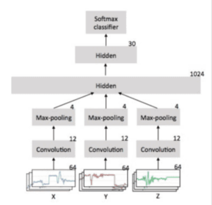
Skip the Academics. Just Results.

SEE WHAT'S INSIDE

Tweet Tweet Share Share

More On This Topic

A Gentle Introduction to Deep Learning for Face Recognition

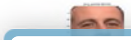


Deep Learning Models for Human Activity Recognition

Never miss a tutorial:

Picked for you:

How to Train an Object Detection Model with Keras



How to Develop a Face Recognition

Start Machine Learning

×

You can master applied Machine Learning **without math or fancy degrees**.
Find out how in this *free* and *practical* course.

Email Address

□ I consent to receive information about services and special offers by email. For more information, see the Privacy Policy.

START MY EMAIL COURSE

Loving the Tutorials?

The Deep Learning for Computer Vision EBook is where you'll find the ***Really Good*** stuff.

>> SEE WHAT'S INSIDE

Start Machine Learning

A Gentle Introduction to the Promise of Deep…

How to Train an Object Detection Model with Keras

9 Applications of Deep Learning for Computer Vision

How to Use Mask R-CNN in Keras for Object Detection…

About Jason Brownlee

Jason Brownlee, PhD is a machine learning specialist who teaches developers how to get results with modern machine learning methods via hands-on tutorials.

View all posts by Jason Brownlee →

‹ Multi-Label Classification of Satellite Photos of the Amazon Rainforest

How to Use Mask R-CNN in Keras for Object Detection in Photographs ›

89 Responses to *A Gentle Introduction to Object Recognition With Deep Learning*

Anna May 22, 2019 at 6:22 pm #

REPLY

Amazing. Thanks for sharing

Never miss a tutorial:

Picked for you:

How to Train an Object Detection Model with Keras

How to Develop a Face Recognition

Start Machine Learning

×

You can master applied Machine Learning **without math or fancy degrees**.
Find out how in this *free* and *practical* course.

Email Address

□ I consent to receive information about services and special offers by email. For more information, see the Privacy Policy.

START MY EMAIL COURSE

Loving the Tutorials?

The Deep Learning for Computer Vision EBook is where you'll find the ***Really Good*** stuff.

>> SEE WHAT'S INSIDE

Start Machine Learning

Jason Brownlee May 23, 2019 at 5:56 am # REPLY ↰

Thanks, I’m glad you found it useful.

Ravin May 29, 2019 at 8:42 pm # REPLY ↰

Hi Jason,

Ive got an “offline” video feed and want to identify objects in that “offline” video feed.
The camera always will be at a fixed angle.
For me accuracy is of utmost importance, can you pls suggest which algorithm will work for me ?

Jason Brownlee May 30, 2019 at 9:00 am # REPLY ↰

Mask RCNN.

Navdeep Singh June 18, 2019 at 1:05 am # REPLY ↰

hi ravin, I gets an 6000 videos daily to detect person, check format and background color and detect logo, how we can do stuff at offline without playing. how did you achieve. @jason you can also guide me . also on architecture of same

Jason Brownlee June 18, 2019 at 6:40 am # REPLY ↰

That is a lot of video!

Perhaps start with simple/fast methods and see how far they get you.

Jian June 22, 2019 at 1:22 am # REPLY ↰

Hi,

I read that FCNs can do pixel level classification, so I’m wondering can FCNs be used to do pixel level regression?

Thanks!

Jason Brownlee June 22, 2019 at 6:45 am # REPLY ↰

I don’t see why not.

naimath August 20, 2019 at 10:57 am # REPLY ↰

Never miss a tutorial:

Picked for you:

How to Train an Object Detection Model with Keras

How to Develop a Face Recognition

Loving the Tutorials?

The Deep Learning for Computer Vision EBook is where you'll find the ***Really Good*** stuff.

>> SEE WHAT'S INSIDE

Start Machine Learning

I would like to check whether parking lot available or camera feed vedio. what should I check ?

Jason Brownlee August 20, 2019 at 2:12 pm #

REPLY

Good question.

Perhaps model as object detection, at least as a starting point?

Ahmad October 9, 2019 at 5:26 pm #

REPLY

Jason,

This material is really great. This gave me a better idea about object localisation and classification. Here I am mentioning all the points that I understood from the blog with respect to object detection.
1. The object detection framework initially uses a CNN model as a feature extractor (Examples VGG without final fully connected layer).
2. This output of the VGG is given to another CNN model known as RPN, which gives a set of areas where potential objects may exists
3. Based on the RPN output, another CNN model (typically a classifier) process the VGG output and gives final results (Object classes and respective bounding boxes)

Now I would like to know what type of CNN combinations are popular for single class object detection problem. If I want to develop a custom model, what are the available resources.

Jason Brownlee October 10, 2019 at 6:52 am #

REPLY

A RCNN or a YOLO would be a great place to start.

regotto October 11, 2019 at 12:36 am #

REPLY

greatly NB

Jason Brownlee October 11, 2019 at 6:22 am #

REPLY

Thanks!

Div October 29, 2019 at 1:28 am #

REPLY

Hey, great article!
I have to make a project where I have to detect and count the number of people in an image which is captured through an Android mobile within a short time. How do I do it?

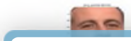
Jason Brownlee October 29, 2019 at 5:28 am #

REPLY

Never miss a tutorial:

Picked for you:

How to Train an Object Detection Model with Keras



How to Develop a Face Recognition

Perhaps start with a data set of images with a known count of people in the image.

Then perhaps test a suite of object detection methods to see what works best on your dataset?

Raz November 3, 2019 at 3:20 am # REPLY ↰

Hey
It's a great article and gave me good insight.
I need to detect the yaw, pitch and roll of cars in addition to their x,y,z position in images from a street. Beside that, I already have mask for the images that show me where are the cars in the image, so I don't think I need to localize the cars in the image. My question is, can I use R-CNN or YOLO to predict the yaw, pitch and roll of cars in the image (of course, those that are not covered with the mask and are reasonably close to the camera that is taken the image)

Jason Brownlee November 3, 2019 at 6:02 am # REPLY ↰

Thanks.

I think you need another model that takes the image input and predicts the coordinate outputs.

Nick December 16, 2019 at 12:01 pm # REPLY ↰

Hi Jason

I would like to track cyclists riding around a Velodrome. There are 7 cyclists in a race all with different colours.

1. What framework would you use?
2. The predominant feature is colour, would you create 7 classes based on each colour?

Jason Brownlee December 16, 2019 at 1:37 pm # REPLY ↰

Great question, I think some research (what is similar/has been tried before) and prototyping (what works) would be a good idea.

Anirudh January 14, 2020 at 6:02 pm # REPLY ↰

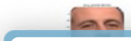
Another Excellent Article Dr. Brownlee,.
I'm confused in the part of the YOLOv1 where the paper's author mentions that the final layer uses a linear activation function. But the outputs are supposed to be between 0 to 1 for all the x,y and w,h and the confidence of the bounding box. And the output also predicts one of twenty classes. Normally, we use softmax for the classification of classes. But it's bot mentioned in the paper if they use it or not. And my intuition is to use sigmoid for the x,y and w,h prediction as they have values between 0 to 1. Even that isn't mentioned anywhere in the paper. If they're not using sigmoid or softmax, then how does the classification process works.

Jason Brownlee January 15, 2020 at 8:21 am # REPLY ↰

Never miss a tutorial:

Picked for you:

How to Train an Object Detection Model with Keras



How to Develop a Face Recognition

Thanks!

Good question. Perhaps check the official source code and see exactly what they did?

Anirudh S January 15, 2020 at 5:23 pm #

REPLY ↰

I went through one of the tensorflow ports of the original darknet implementation. And it seems to just produce linear outputs and couldn't find any sigmoid or softmax. Wouldn't that be a little more unconstrained since they have to predict a value between 0 and 1 but they're predicted value doesn't have any bounds as it's linear? Also, if YOLO predicts one of the twenty class probabilities and confidence with a linear function, that seems more confusing!
But the paper says " We normalize the bounding box width and height by the image width and height so that they fall between 0 and 1. We parametrize the bounding box x and y coordinates to be offsets of a particular grid cell location so they are also bounded between 0 and 1."
How do they bound the values between 0 and 1 if they're not using a sigmoid or softmax?

Jason Brownlee January 16, 2020 at 6:09 am #

REPLY ↰

Perhaps the quote from the paper has to do with the preparation of the training data for the model.

BRIAN HARRAH January 16, 2020 at 7:36 am #

REPLY ↰

Jason, noob question: When training a model with tagged images, does the algorithm only concern itself with the content that's inside the human-drawn bounding box(es)? Or does it still use the content that lies outside the bounding boxes as well? Perhaps this varies with the type of model you are training and/or the method you use to train it?

Any help is greatly appreciated.

Jason Brownlee January 16, 2020 at 1:32 pm #

REPLY ↰

Great question!

The model sees the whole image and the bounding box. It learns where to put the box in the image – what is in and what is out.

Brian Harrah January 17, 2020 at 2:20 am #

REPLY ↰

Thanks for the reply! So if the model is training with the whole image, would the resulting prediction model be more accurate if the training images were "cropped" in such a way as to remove as much of the area outside the bounding box as possible? In other words, training the model with essentially only what lies inside the box that we want to detect.

Jason Brownlee January 17, 2020 at 6:04 am #

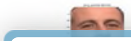
REPLY ↰

Correct.

Never miss a tutorial:

Picked for you:

How to Train an Object Detection Model with Keras



How to Develop a Face Recognition

Loving the Tutorials?

The Deep Learning for Computer Vision EBook is where you'll find the ***Really Good*** stuff.

>> SEE WHAT'S INSIDE

Start Machine Learning ^

Sapan February 14, 2020 at 4:40 pm #

REPLY ↰

Hi Jason,

Really informative article!

I had a question related to this. What model should I use if I want to detect an object that is tilted in any direction, i.e. it is not in the same upright vertical position as the image is. May be tilted at random angles in all different images.

Also, I need to get the coordinates of center of that object. Any pre-trained model that could help here??

Thank you

Jason Brownlee February 15, 2020 at 6:23 am #

REPLY ↰

Thanks!

Same types of models, although trained to expect these transforms. E.g. data augmentation would be helpful.

Maswood Ahmed April 7, 2020 at 12:34 pm #

REPLY ↰

Brownlee sir, Really its a amazing. Sir I want to know about Mask R-CNN . can I use it to develop my Mtech project 'face detection and recognition" , sir please help me in this regard. waiting for your reply eagerly.

Jason Brownlee April 7, 2020 at 1:31 pm #

REPLY ↰

Thanks!

I don't recommend mask rcnn for face recognition, use mtcnn + facenet or vggface2:
https://machinelearningmastery.com/how-to-develop-a-face-recognition-system-using-facenet-in-keras-and-an-svm-classifier/

Jasraj Singh April 13, 2020 at 9:32 pm #

REPLY ↰

Hi

After discussing the ILSVRC paper, the article says, "Single-object localization: Algorithms produce a list of object categories present in the image, along with an axis-aligned bounding box indicating the position and scale of one instance of each object category."
Isn't the localization process just supposed to be about producing a boundary for the object? Does it also classify the object in a category? Or is this the definition for 'Single-object detection' instead?

Jason Brownlee April 14, 2020 at 6:16 am #

REPLY ↰

Yes, typically classify and draw a box around the object.

Never miss a tutorial:

Picked for you:

How to Train an Object Detection Model with Keras

How to Develop a Face Recognition

Start Machine Learning

×

You can master applied Machine Learning **without math or fancy degrees**.
Find out how in this *free* and *practical* course.

Email Address

□ I consent to receive information about services and special offers by email. For more information, see the Privacy Policy.

START MY EMAIL COURSE

Loving the Tutorials?

The Deep Learning for Computer Vision EBook is where you'll find the ***Really Good*** stuff.

>> SEE WHAT'S INSIDE

Start Machine Learning ^

Jeetendra Dhall March 21, 2021 at 9:19 pm #

REPLY ↰

The definitions of Single-object localization and Object Detection are having the same text. I believe this is a typo. Single-object localization should not have classification. It should only have the bounding rectangle, as Jasraj suggests.

Jason Brownlee March 22, 2021 at 5:30 am #

REPLY ↰

Thanks for sharing.

nbro June 14, 2020 at 5:24 am #

REPLY ↰

You say "divided into a 7×7 grid and each cell in the grid may predict 2 bounding boxes, resulting in 94 proposed bounding box predictions", so that means there will be 7*7=49 cells. If each cell predicts two bounding boxes, then the number of proposals is 49*2 = 98 (and not 94).

Mohamad June 14, 2020 at 10:03 pm #

REPLY ↰

Hello Jason,

thanks you very much for the article, fantastic like always. I have a question please:

Regarding the RPN (during training) in Faster RCNN, what happen if the image being processed has no ground truth boxes (background image), do we still get anchor boxes as well even though we don't have the GT boxes, or does the whole image get recorded as a negative example in this case?

I will be glad for any help (:

Jason Brownlee June 15, 2020 at 6:06 am #

REPLY ↰

You're welcome.

If you don't have bounding boxes in the training data, you cannot train an object detection model.

Som Dubey June 15, 2020 at 11:12 pm #

REPLY ↰

Very informative read. Thanks a lot.
I have a query regarding YOLO1. Its researched paper says –
"Our system divides the input image into an S × S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object".
I am not able to understand that for new and unseen images (live a live video feed), how algorithm is able to find out where exactly objects are present in the picture and thus their center? Also, in the real time scenario, there will not be any Ground truth to have comparison with, how it finds out IoU and thus the respective probability of having an object in a box.

Jason Brownlee June 16, 2020 at 5:39 am #

REPLY ↰

Never miss a tutorial:

Picked for you:

How to Train an Object Detection Model with Keras

How to Develop a Face Recognition

Start Machine Learning

×

You can master applied Machine Learning **without math or fancy degrees**.
Find out how in this *free* and *practical* course.

Email Address

□ I consent to receive information about services and special offers by email. For more information, see the Privacy Policy.

START MY EMAIL COURSE

Loving the Tutorials?

The Deep Learning for Computer Vision EBook is where you'll find the ***Really Good*** stuff.

>> SEE WHAT'S INSIDE

Start Machine Learning

Thanks.

Perhaps this worked example will help:
https://machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/

Sutharsan July 1, 2020 at 12:22 am #

REPLY

Hi Jason,

This is a great article to get some ideas about the algorithms since I'm new to this area. I'm an final year undergraduate currently working on a research topic "Vehicle Detection in Satellite Images". I would like to know which algorithm can be used or works better for the topic.

Thank you.

Jason Brownlee July 1, 2020 at 5:53 am #

REPLY

I recommend testing a suite of algorithms and configurations on your dataset in order to discover what works best.

It is a good idea to start with transfer learning based approaches.

Sophie March 3, 2021 at 9:00 pm #

REPLY

Hello Sutharsan and Jason,

I am also currently working on the same topic "Vehicle detection in satellite images" and I would like to know what algorithm did you find better.

Jason Brownlee March 4, 2021 at 5:48 am #

REPLY

I recommend testing a suite of different methods in order to discover what works well or best for your specific dataset.

Junior July 22, 2020 at 6:57 am #

REPLY

Great article! I was confused about the terminology of object detection and I think this article is the best about it. I wanted to ask you, I'm using MobileNetV2 for object detection, but after reading this I'm not sure if that was the correct choice. I need something fast for predictions due to we need this to work on CPU, now we can predict at a 11 FPS, which works well for us, but the bounding box predicted is not oriented and that complicate things a little. What would you recommend to use to have similar FPS (or faster) and a similar accuracy or at least an oriented bounding box? I was thinking in using landmarks but I don't know if that will suit our needs.

Thanks for sharing your knowledge!

Jason Brownlee July 22, 2020 at 7:37 am #

REPLY

Perhaps test a suite of models and see which best meets your specific speed requirements.

Never miss a tutorial:

Picked for you:

How to Train an Object Detection Model with Keras

How to Develop a Face Recognition

ts

e)

Loving the Tutorials?

The Deep Learning for Computer Vision EBook is where you'll find the ***Really Good*** stuff.

>> SEE WHAT'S INSIDE

Start Machine Learning

Abdullah August 5, 2020 at 4:45 pm # REPLY ↰

Hello dear, My name is Abdullah and I want to do research on object recognition/classification. I have done my master's degree in Mathematics 2018. Now I turning here and want to do research in object recognition/classification with major mathematics. Can you suggest to me where I have to go? I am making a research proposal in object recognition/classification with my strength in mathematics. Can you please help me???

Thanking you

Jason Brownlee August 6, 2020 at 6:10 am # REPLY ↰

Sorry, I cannot help you with a research proposal.

Mehnaz Tabassum August 7, 2020 at 11:41 am # REPLY ↰

Dear Author,
It's an informative article indeed. Could you please help me giving the information that in this pipeline where is the place of "Object Proposal"? I am a little bit confused about object localization and object proposal.
Thanks in advance

Jason Brownlee August 7, 2020 at 1:30 pm # REPLY ↰

Thanks.

I believe "proposals" are candidate predictions.

prachi October 11, 2020 at 2:04 am # REPLY ↰

DEAR SIR

IT IS VERY INFORMATIVE ARTICLE. Can you pls help in giving the information that in text detection in natural images which alogorithm works well and about the synthetic images . i am little bit confused.

Jason Brownlee October 11, 2020 at 6:53 am # REPLY ↰

Thanks.

Thanks for the suggestion, I hope to write about that topic in the future.

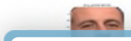
Joseph A Urban October 13, 2020 at 3:37 pm # REPLY ↰

What if an MV system is in a room and can detect a window, door and ceiling lamp, and it can match it up with a pre-defined set of the same objects whose attributes include each object's identification and position in that same room. Shouldn't a separate system able to extrapolate the vectors to each of those

Never miss a tutorial:

Picked for you:

How to Train an Object Detection Model with Keras



How to Develop a Face Recognition

Start Machine Learning ✕

You can master applied Machine Learning **without math or fancy degrees**.
Find out how in this *free* and *practical* course.

Email Address

□ I consent to receive information about services and special offers by email. For more information, see the Privacy Policy.

START MY EMAIL COURSE

Loving the Tutorials?

The Deep Learning for Computer Vision EBook is where you'll find the ***Really Good*** stuff.

>> SEE WHAT'S INSIDE

Start Machine Learning ^

objects and determine its location relative to everything else in the room. Is there a name for this pre-defined framework reference?

Jason Brownlee October 14, 2020 at 6:10 am # REPLY ↰

Sorry, I don't don't know of models that can do what you describe. It sounds like a "system" (software package) not a single model.

Derek October 14, 2020 at 9:06 pm # REPLY ↰

Hello, thanks for the very informative article (like yours always are). I have a dataset of powerpoint slides and need to build a model to detect for logos in the slides. The dataset has labels for the presence of logos y={0,1}. Which model would you recommend? Thank you

Jason Brownlee October 15, 2020 at 6:08 am # REPLY ↰

Perhaps test a suite of models and discover what works best for your specific dataset.

Kaustubh Sharma October 16, 2020 at 6:44 am # REPLY ↰

Thanks for the simple yet detailed article and explanation. I was wondering if there is a way to get bounding boxes with older models like VGG16?

Jason Brownlee October 16, 2020 at 8:09 am # REPLY ↰

VGG16 is only for feature extraction and classifying images.

KUMAR November 15, 2020 at 6:16 pm # REPLY ↰

HELLO SIR, FOR DOING PROJECT ON OBJECT RECOGNITION WHAT ARE THE THINGS WE HAVE TO LEARN AND IS THERE ANY BASIC PAPERS TO STUDY …. I THINK YOUR REPLY WILL BE HELPFUL.

THANK YOU SIR.

Jason Brownlee November 16, 2020 at 6:25 am # REPLY ↰

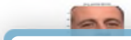
I recommend this book:
https://machinelearningmastery.com/deep-learning-for-computer-vision/

KUMAR November 16, 2020 at 3:21 pm # REPLY ↰

Never miss a tutorial:

Picked for you:

How to Train an Object Detection Model with Keras



How to Develop a Face Recognition

Loving the Tutorials?

The Deep Learning for Computer Vision EBook is where you'll find the ***Really Good*** stuff.

>> SEE WHAT'S INSIDE

Start Machine Learning

In that book can we get all the information regarding the project (object recognition) and can you please suggest the best courses for python and deep learning so that i will get enough knowledge to do that project(object recognition)

thank you sir.

Jason Brownlee November 17, 2020 at 6:25 am # REPLY

The book provides examples of object detection and how to apply a pre-trained object detection model and how to train a model for a new dataset.

It is a great place to start.

Otherwise, you can see the free tutorials here:
https://machinelearningmastery.com/start-here/#dlfcv

SURESH November 18, 2020 at 2:24 pm # REPLY

sir, suggest me python course for data science projects(ML,DL)?

Jason Brownlee November 19, 2020 at 7:41 am # REPLY

See this:
https://machinelearningmastery.com/faq/single-faq/what-machine-learning-project-should-i-work-on

SURESH November 18, 2020 at 2:58 pm # REPLY

Sir ,

I want to know the history of object recognition, i.e when it was started , what are the algorithms used and what are the negatives ? is it available anywhere?

Jason Brownlee November 19, 2020 at 7:41 am # REPLY

Perhaps you can find a few review papers that provide this literature survey.

I recommend searching on scholar.google.com

Octavio December 5, 2020 at 5:03 am # REPLY

Great article, Really informative, thank you for sharing.

I'm making a light-weight python based platform for interfacing and controlling
any (supported) devices (mostly for people learning to code, and those that want a framework for automation without having to go through the pain learning how to communicate with said device).
I am in the process of building some tools that would help people perform more interesting programs / bots with these devices one of which is processing

captured images.
As I want this to be simple and rather generic, the users currently make two directories, one of images that they want to detect, and one of images that they want to ignore, training/saving the model is taken care of for them. then they can detect the centers of instances of the images they want in a larger images. at the moment I'm using a cnn and then I test various sub images in the larger one hence creating a probability heatmap, and finish the task by finding the peaks which indicate the locations of images the cnn finds. (currently all the sub images take a while (~0.5-1s) to process.
Do you think it would be possible to use an RCNN to perform this task whilst keeping the simplicity similar i.e. somehow avoid the user having to create bounding box datasets?

One idea I had was to plant the training dataset into larger images randomly, this way I would have the bounding box and could train the rcnn on these larger images? Your thoughts would be greatly appreciated.

Thank you so much for the article.

Jason Brownlee December 5, 2020 at 8:13 am # REPLY ↰

Thanks.

Hard to say, perhaps develop a prototype and test your ideas.

Chandu Krishna December 28, 2020 at 4:30 am # REPLY ↰

Hi Jason,

I'm currently working on data annotation i.e object detection using bounding boxes and also few projects such as weather conditions , road conditions for autonomous cars. I want to upgrade myself to the next process(what's the next step and annotating the objects) could you please help what course if I learn I can go more deep into the autonomous cars field. I learnt something different from your article regarding object detection, please suggest me what to do to improve my job skills.

Thanks for your article

Jason Brownlee December 28, 2020 at 6:03 am # REPLY ↰

Thanks, sorry, I don't have many tutorials on object detection. I hope to write more on the topic in the future.

chandu Krishna December 28, 2020 at 4:39 pm # REPLY ↰

Thanks for your response Jason, to continue in the ADAS field if I learn Machine Learning will it be a good move for my future?

Jason Brownlee December 29, 2020 at 5:10 am # REPLY ↰

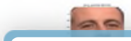
What is "ADAS"?

Only you know what will be good for you future.

Never miss a tutorial:

Picked for you:

How to Train an Object Detection Model with Keras



How to Develop a Face Recognition

Guojiang January 26, 2021 at 11:26 pm # REPLY ↰

A confusing typing error in YOLO: 'if the center of a bounding box falls within it', it should be the center of an object.

Jason Brownlee January 27, 2021 at 6:08 am # REPLY ↰

Thanks, fixed.

Surya February 5, 2021 at 1:28 am # REPLY ↰

Hi jason – If I have to draw a bounding box of same class at different locations of the image. Which algorithm should I go for and why?

Jason Brownlee February 5, 2021 at 5:42 am # REPLY ↰

Most object recognition models support multiple objects in the image.

Perhaps try a few and discover what works well or best for your dataset.

Rahul Kumar March 23, 2021 at 10:39 pm # REPLY ↰

Very informative…

Jason Brownlee March 24, 2021 at 5:51 am # REPLY ↰

Thanks!

frans April 8, 2021 at 12:33 am # REPLY ↰

Hi,

for a museum, we would like to be be able if one of 20 specific objects is placed anywhere on a table.
Our original line of thought went from "user selects the object on a touch screen", to "super expensive touch table that recognizes tags", to "detect one of twenty nfc/rfid tags placed on reader", to what I'm wondering about now "use deeplearning to recognize the object any where on the table"
I have however no clue if that is feasible, and what is best and how to proceed.

Jason Brownlee April 8, 2021 at 5:10 am # REPLY ↰

Perhaps you can try to develop a small prototype to see if it is feasible for your problem/data.

KKarlovic April 11, 2021 at 3:51 am #

REPLY

Look at my robotic arm that detects an object and then sorts the object by the number of holes. See the complete documentation and code here: https://github.com/k-karlovic/robotic-object-sorter-with-computer-vision

Jason Brownlee April 11, 2021 at 4:56 am #

REPLY

Thanks for sharing.

snehal July 28, 2021 at 7:25 pm #

REPLY

Hi

I did object detection using YOLOv5. Now i want to apply reinforcement learning approach on that detected images. Is there any way to do that? Is there any solution related this? please help me.

Jason Brownlee July 29, 2021 at 5:10 am #

REPLY

Sorry, I don't have any examples of RL, I cannot give you good off the cuff advice on the topic.

Ragavee S.V August 13, 2021 at 3:57 pm #

REPLY

Can I get the predicted labels as a list ?(Giving a new image to the detection model and extracting the predicted labels as a list) Like if the test image contains objects of class 'person' and 'dog' , list should contain ['person' , 'dog']
If yes please share me the code.
Thanks in advance.

Adrian Tam August 14, 2021 at 2:49 am #

REPLY

Surely you can. I am not sharing the code but the idea is, usually the prediction is in terms of probability to each label and we select the label as the output. That should be a piece of code **outside** of the model to prepare these label and there is where you can make that happen.

Jokonug January 7, 2022 at 6:04 pm #

REPLY

Hi Jason,
wonder if you can guide me a little bit.

I started new project to deliver people detection however in certain area only (not in the frame of the picture or video). The intention is to warn or give alarm if people entering line-of-fire area, in particulary below crane operating area.
some how, the area is dynamic event by event so the boundary some is not fix. So I wonder if we do virtual boundary using some of helpful object (i.e. briht and colorful cone) and then make a segmentation and then object detection. if those two overlap then raise alarm.

Never miss a tutorial:

Picked for you:

How to Train an Object Detection Model with Keras

How to Develop a Face Recognition

Start Machine Learning

×

You can master applied Machine Learning **without math or fancy degrees**.
Find out how in this *free* and *practical* course.

Email Address

□ I consent to receive information about services and special offers by email. For more information, see the Privacy Policy.

START MY EMAIL COURSE

Loving the Tutorials?

The Deep Learning for Computer Vision EBook is where you'll find the ***Really Good*** stuff.

>> SEE WHAT'S INSIDE

Start Machine Learning

Do you have tutorial or your e-book that cover that thing (especially virtual boundary)?
Thanks

James Carmichael January 8, 2022 at 11:11 am #

REPLY

Interesting application Jokonug! While I cannot speak directly to your application, I would highly recommend that you acquire the following ebook:

https://machinelearningmastery.com/deep-learning-for-computer-vision/

Leave a Reply

Name (required)

Email (will not be published) (required)

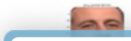
SUBMIT COMMENT

Welcome!
I'm *Jason Brownlee* PhD
and I **help developers** get results with **machine learning**.
Read more

Never miss a tutorial:

Picked for you:

How to Train an Object Detection Model with Keras



How to Develop a Face Recognition

Loving the Tutorials?

The Deep Learning for Computer Vision EBook is where you'll find the ***Really Good*** stuff.

>> SEE WHAT'S INSIDE

Start Machine Learning

Never miss a tutorial:

Picked for you:

How to Train an Object Detection Model with Keras

How to Develop a Face Recognition

Loving the Tutorials?

The Deep Learning for Computer Vision EBook is where you'll find the ***Really Good*** stuff.

>> SEE WHAT'S INSIDE

Start Machine Learning

LinkedIn | Twitter | Facebook | Newsletter | RSS

Privacy | Disclaimer | Terms | Contact | Sitemap | Search